

**SINUMERIK 840D sl**

**SINUMERIK Integrate  
Run MyVNCK  
Reference**

**Function Manual**

<b>General Information</b>	<b>1</b>
<b>System Characteristics</b>	<b>2</b>
<b>Interface Functional Description</b>	<b>3</b>
<b>VPLC Processing</b>	<b>4</b>
<b>NCU Link Processing</b>	<b>5</b>
<b>VNCK License</b>	<b>6</b>
<b>More General Information</b>	<b>7</b>




**Valid for**

Control SINUMERIK 840D sl	<i>Version</i> 4.5 SP2
Software SINUMERIK Integrate Run MyVNCK	4.5 SP1

## Legal information

### Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

 <b>DANGER</b>
indicates that death or severe personal injury <b>will</b> result if proper precautions are not taken.
 <b>WARNING</b>
indicates that death or severe personal injury <b>may</b> result if proper precautions are not taken.
 <b>CAUTION</b>
with a safety alert symbol, indicates that minor personal injury can result if proper precautions are not taken.
<b>CAUTION</b>
without a safety alert symbol, indicates that property damage can result if proper precautions are not taken.
<b>NOTICE</b>
indicates that an unintended result or situation can occur if the relevant information is not taken into account.


If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

### Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

### Proper use of Siemens products

Note the following:

 <b>WARNING</b>
Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

### Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

### Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

# Preface

## **SINUMERIK documentation**

The SINUMERIK documentation is organized in the following categories:

- General documentation
- User documentation
- Manufacturer/service documentation

## **Additional information**

You can find information on the following topics under the link [www.siemens.com/motioncontrol/docu](http://www.siemens.com/motioncontrol/docu):

- Ordering documentation/overview of documentation
- Additional links to download documents
- Using documentation online (find and search in manuals/information)

If you have any questions regarding the technical documentation (e.g. suggestions, corrections) then please send an e-mail to the following address:  
<mailto:docu.motioncontrol@siemens.com>

## **My Documentation Manager (MDM)**

Under the following link you will find information to individually compile OEM-specific machine documentation based on the Siemens content:  
MDM [www.siemens.com/mdm](http://www.siemens.com/mdm)

## **Training**

For information about the range of training courses, refer under:

- SITRAIN [www.siemens.com/sitrain](http://www.siemens.com/sitrain) - training courses from Siemens for products, systems and solutions in automation technology
- SinuTrain [www.siemens.com/sinutrain](http://www.siemens.com/sinutrain) - training software for SINUMERIK

## **FAQs**

You can find Frequently Asked Questions in the Service&Support pages  
Product Support [www.siemens.com/automation/service&support](http://www.siemens.com/automation/service&support)

## **SINUMERIK**

You can find information on SINUMERIK under the following link:  
[www.siemens.com/sinumerik](http://www.siemens.com/sinumerik)

**Target group**

This publication is intended for project engineers, programmers, technologists (of machine manufacturers), and system startup engineers (of systems/machines).

**Benefits**

The Function Manual describes the functions so that the target group is familiar with and can select them. It provides the target group with the information required to implement the functions.

Utilization phase: Planning and configuration phase, implementation phase, setup and commissioning phase

**Standard version**

Extensions or changes made by the machine manufacturer are documented by the machine manufacturer.

Other functions not described in this documentation might be executable in the control. This does not, however, represent an obligation to supply such functions with a new control or when servicing.

Further, for the sake of simplicity, this documentation does not contain all detailed information about all types of the product and cannot cover every conceivable case of installation, operation or maintenance.

**Technical Support**

You can find telephone numbers for other countries for technical support in the Internet under "Contact" [www.siemens.com/automation/service&support](http://www.siemens.com/automation/service&support).

**SINUMERIK Internet address**

<http://www.siemens.com/sinumerik>

# Contents

<b>1 General Information.....</b>	<b>9</b>
1.1 VNCK Version History .....	9
1.2 VNCK Architecture.....	10
1.3 Installation and Deinstallation .....	12
1.4 Installation Folders.....	13
1.5 Silent Installation.....	15
1.6 Documentation.....	16
1.7 Additional Documentation for Developers .....	17
<b>2 System Characteristics .....</b>	<b>19</b>
2.1 Chronological Sequence of Program Processing.....	19
2.2 Asynchronous Communication Simulation <-> VNC Server .....	20
2.3 Interfaces .....	21
2.3.1 Single NCU Processing .....	21
2.3.2 VPLC Processing.....	22
2.3.3 NCU Link Processing.....	23
2.3.4 License Handling .....	24
2.4 Interface Function Return Values .....	25
2.5 Freeze Handling in Simulation.....	25
2.6 Creating high performance traces .....	25
<b>3 Interface Functional Description.....</b>	<b>27</b>
3.1 VNCK Boot and Shutdown .....	27
3.1.1 General Information.....	27
3.1.2 Establishing the connection between controller and simulation.....	27
3.1.3 Controller start-up.....	28
3.1.4 Controller Status Saving.....	31
3.1.5 Controller Status Refreshing .....	33
3.1.6 Resetting the VNCK Kernel .....	34
3.1.7 Controller Shutdown .....	35
3.1.8 Initializing the Channel Axes Values .....	36
3.1.9 Setting VNCK System Sleeptime .....	37
3.1.10 Setting the VNCK Kernel Process State.....	38
3.2 VNCK Configuration services .....	39
3.2.1 General Information.....	39

3.2.2	Retrieving the NC configuration.....	39
3.2.3	Retrieving the axes configuration .....	40
3.2.4	Retrieving the VNCK Server Version .....	41
3.3	Domain Data Management.....	42
3.3.1	General Information.....	42
3.3.2	Retrieving existing project directories.....	43
3.3.3	Retrieving a program list within a project directory.....	44
3.3.4	Transferring a project to the VNCK .....	45
3.3.5	Transferring a program to a project directory .....	46
3.3.6	Transferring a setting data file to VNCK.....	47
3.3.7	Copying a program from VNCK to a local folder .....	48
3.3.8	Deleting a file in VNCK data management system.....	49
3.3.9	Obtaining information about multi-file transfer.....	50
3.3.10	Obtaining information about transfer status .....	50
3.4	Variable Services .....	51
3.4.1	General Information.....	51
3.4.2	Reading Variables .....	52
3.4.3	Writing Variables .....	53
3.4.4	Watching Variables.....	54
3.4.5	Reading BTSS Variables.....	56
3.5	VDI Services .....	57
3.5.1	General Information.....	57
3.5.2	Writing to the VDI Interface .....	57
3.5.3	Reading from the VDI Interface.....	58
3.5.4	Handling FastIO via VDI Variables.....	59
3.5.5	Handling TSM mask in SINUMERIK Operate .....	59
3.6	VDI Fast Input and Output Services .....	62
3.7	VNCK Program Control by Slices .....	64
3.7.1	General Information.....	64
3.7.2	Setting slice mode .....	67
3.7.3	Setting freeze mode .....	68
3.7.4	Processing the next slice.....	68
3.7.5	Controller freeze .....	69
3.7.6	Timer Functions.....	69
3.8	NC Program Control Services .....	71
3.8.1	Program selection.....	71
3.8.2	Program selection for external processing .....	72
3.8.3	Enabling program execution.....	73
3.8.4	Stopping program execution.....	74
3.8.5	Resetting program execution.....	75
3.9	Extended Program Control Services .....	77
3.9.1	General Info.....	77
3.9.2	Registering Patterns for Interpretation .....	77
3.9.3	Executing registered NC commands.....	79
3.9.4	Enabling Path Interpolation .....	81
3.10	Path Data Services .....	82
3.10.1	Setting path data output option .....	82

3.10.2	Getting path data output events .....	84
3.10.3	Sending path output data .....	84
3.10.4	Handling Collision Limits .....	85
3.10.5	Setting Actual MCS Axes Positions .....	86
3.11	Program Data Services .....	87
3.11.1	Setting program data output option .....	87
3.11.2	Program display .....	88
3.11.3	User program message .....	88
3.11.4	Beginning of a new motion .....	89
3.11.5	Single Axis Motion Management .....	91
3.11.6	IPO Block Change .....	92
3.11.7	Tool selection .....	93
3.11.8	Tool change .....	94
3.11.9	Selecting a new tool offset .....	95
3.11.10	ToolCarrier Selection .....	96
3.11.11	Subroutine call .....	96
3.11.12	Return from subroutine .....	97
3.11.13	Workpiece .....	97
3.11.14	Fixture .....	98
3.11.15	Fixed Stop and Measurement .....	99
3.12	Alarm Management .....	103
3.12.1	Alarm occurred .....	103
3.12.2	Alarm deleted .....	104
3.12.3	Cancelling alarms .....	105
3.13	OEM Compile Cycles .....	106
3.14	Extended Services .....	106
3.15	EXTCALL .....	106
3.16	Reading / Writing values of Initial Parameters .....	107
3.16.1	General Information .....	107
3.16.2	Writing the value of an initial parameter .....	108
3.16.3	Reading the value of an initial parameter .....	109
<b>4</b>	<b>VPLC Processing .....</b>	<b>111</b>
4.1	VPLC Initializing and Shutdown .....	112
4.1.1	Establishing the Controller – Simulator Connection .....	112
4.1.2	Initializing VPLC Handling .....	112
4.1.3	Providing VPLC IO Hardare Configuration .....	113
4.1.4	Terminating VPLC Handling .....	113
4.1.5	Controlling CPU usage of the VPLC process .....	113
4.2	VPLC Leds and Switches .....	114
4.2.1	Reading VPLC Operation State .....	114
4.2.2	Watching VPLC Operation State .....	114
4.2.3	Setting VPLC Switch .....	115
4.2.4	Reading VPLC Switch .....	115
4.2.5	Watching VPLC Operation States .....	116
4.3	VPLC Progress Control .....	117

4.3.1 Activating VPLC Synchronisation .....	117
4.3.2 Deactivating VPLC Synchronisation .....	117
4.3.3 VPLC Freeze .....	118
<b>5 NCU Link Processing .....</b>	<b>119</b>
5.1 Link System Initializing and Shutdown .....	120
5.1.1 Establishing the Controller – Simulator Connection .....	120
5.1.2 Initializing the Link System .....	121
5.1.3 Defining the NCU Units .....	122
5.1.4 Controlling the NCU Startups .....	123
5.1.5 Setting the Link State .....	124
5.1.6 Terminating a Link Session .....	125
5.1.7 Link System Startup via a VMF .....	126
5.2 Link System Progress Controlling .....	127
5.2.1 Setting Link Slices .....	127
5.2.2 Processing the next Slice .....	128
5.2.3 Link Freeze .....	129
5.3 Link System Status Saving and Resetting .....	130
5.3.1 Saving the States of the Link Components .....	130
5.3.2 Refreshing the States of the Link Components .....	131
5.4 Link NCU Managemant .....	132
5.4.1 Requiring Simulation Callback Objects .....	132
5.4.2 Establishing the Link NCU Controller – Simulator Connection .....	133
5.4.3 Link NCU Controller start-up .....	134
5.4.4 Setting Link NCU Slice Mode .....	135
5.4.5 Link NCU Controller Shutdown .....	135
5.4.6 Link NCU Controller Freeze .....	136
<b>6 VNCK License .....</b>	<b>137</b>
6.1 ISV License Checking .....	137
<b>7 More General Information .....</b>	<b>139</b>
7.1 Preparation of the HMI Base System .....	139
7.1.1 Enabling the OPC Data Access .....	139
7.1.2 Single NCU Setting .....	140
7.1.3 Link NCU Setting .....	141
7.2 Languages .....	141



# 1 General Information

## 1.1 VNCK Version History

The VNCK system matches to a specific release version of the SINUMERIK software. The following table gives an overview about VNCK version, SINUMERIK release version, the internal NCK version, version of HMI-Advanced and SINUMERIK Operate:

Tabelle 1-1: VNCK Version History

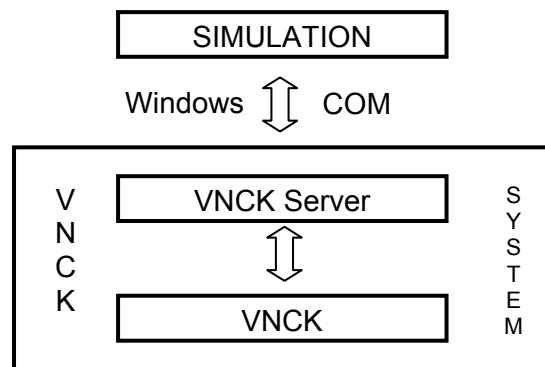
VNCK	SINUMERIK		NCK	HMI-Advanced	SINUMERIK Operate
	powerline (pl)	solutionline (sl)			
1.6	7.4	2.4 1.4	67.05	07.30.23.01	-
1.6 SP1	7.4	1.4 SP1 HF1 2.4 SP1 HF2	67.07.03	07.30.23.04	-
1.6 SP2	7.4	1.4 SP1 HF3 2.4 SP1 HF5	67.07.06	07.30.23.04	-
2.1	7.4	1.5 HF5 2.5 HF2	72.06	07.30.46.00	-
2.6	-	2.6 SP1 HF1	78.05.04	07.50.22.01	02.06.01.01.008
2.6 SP1	-	2.6 SP1 HF3	78.06.03	07.50.22.01	02.06.01.07.002
4.4	-	2.7 SP1 HF3 4.4 SP1 HF3	83.03.07	07.60.59.05	04.04.01.03.001
4.4 SP1	-	2.7 SP2 4.4 SP2	83.04.06	07.60.59.06	04.04.02.00.013
4.5	-	4.5 SP1	87.04.04	07.60.59.06	04.05.01.00.020
4.5 SP1	-	4.5 SP2	87.09	07.60.59.06	04.05.02.00.029

## 1.2 VNCK Architecture

The VNCK system consists of several components. To facilitate understanding the simulator should be aware of the following:

There is a 'VNCK Server' that offers interfaces for all commands to the VNCK. This expression refers to the cross compiled version of a real target SINUMERIK 840D sl system. The server handles operation with this kernel. Thus the simulator need not know anything about the communication system between an HMI system and this kernel. The server makes use of many original SINUMERIK 840D sl HMI servers. This is a simple way of reducing new software development work and ensuring correct handling of tested functionality.

Using Windows COM interfaces the simulator can call the VNC server and transmit commands to the VNCK system. On the other side the VNC server calls interfaces implemented on the simulation side to report the results of given commands as well as data from program processing inside the VNCK.



The following picture gives a rough definition of the VNCK architecture. It shows how the VNCK components communicate with each other and it defines which components are delivered with the VNCK CD.

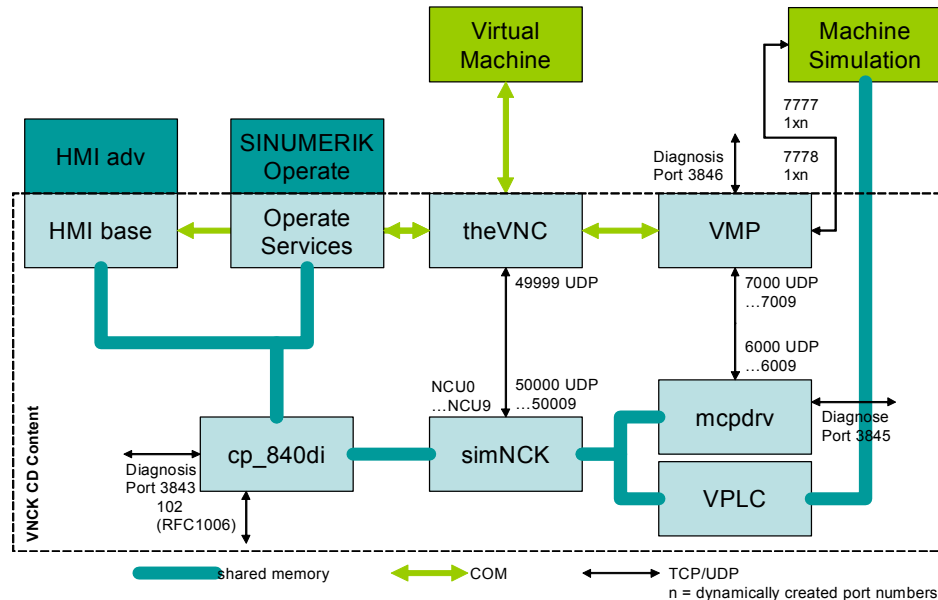


Fig. 1-1:VNCK CD content

Communication base between simNCK and VNCKServer:

The communication between simNCK and VNCKServer was reimplemented in VNCK4.5 based on a combination of shared memory communication (data exchange) and semaphores (event handling). This enhancement removes limitations of older VNCK versions concerning e. g. a running Windows ICS service or runtime problems with antivirus software. The modification is transparent to the user.

If for any reason the old mechanism based on UDP communication needs to be re-established, the following setting must be activated in theVNC.ini:

- [VNCK]
- CCSIM\_COMMUNICATION=0

Removing this statement or using value 1 will enable the new mechanism.

Communication ports:

theVNC:

- Port 49999 is used to communicate between VNCKServer and simNCK instances

simNCK:

- The ports 50000 – 50009 are used to communicate to VNCKServer
- Port 50000 is used with the first instance of simNCK
- The ports 50001 – 50009 are only used if VNCK is used with NCULink
- VNCK4.5 is released for a maximum of three NCUs running in link, thus only the additional ports 50001 and 50002 are used

cp\_840di:

- Port 102 is used to communicate with other CP instances via TCP connection
- Port 3843 is used for diagnostics only (e.g. the ViewLog application)

mcpdrv:

- Port 5000 (not shown in Fig. 1-1) is used to register virtual machine control panels via vmcp.dll
- The ports 6000 – 6009 are reserved for runtime communication between mcpdrv and virtual machine control panel applications. Depending on the availability of these ports the first free port will be used for communication.
- Port 3845 is used for diagnostics only (e.g. the ViewLog application).

VMP:

- The ports 7000 – 7009 are reserved for runtime communication between mcpdrv and virtual machine control panel applications. Depending on the availability of these ports the first free port will be used for communication.
- Port 3846 is used for diagnostics only (e.g. the ViewLog application).

## 1.3 Installation and Deinstallation

Start setup.exe from the VNCK CD. You can choose to install VNCK with runtime components, OEM/ISV components or your custom component definition.

Runtime components are:

- VNCK
- Operate Services
- No VPLC
- No HMIBase services
- No Desktop Shortcuts
- No dongle driver
- No additional files for developers

OEM/ISV components are:

- VNCK
- Operate Services
- Desktop Shortcuts
- Dongle driver
- Additional files for developers
- No VPLC
- No HMIBase services

To uninstall VNCK use 'Start → Settings → Control Panel → Add or Remove Programs' and choose uninstallation of VNCK V04.05.01.00.

## 1.4 Installation Folders

With VNCK version 4.5 the installation folders were modified according to the following list:

<InstallPath>	Installation path of VNCK software  All binaries are stored in this folder. It must not be modified by standard users.
<AllUsersPath>	ISV/OEM specific folder  This folder shall contain license files and ISV/OEM specific versions of theVNC.ini.
<UserPath>	User specific folder  All temporary data of VNCK is stored here (in older versions of VNCK identical to tmp folder in VNCK standard installation.
<HMIAdvPath>	Installation path of HMI-Advanced software
<OperatePath>	Installation path of SINUMERIK Operate software

The standard paths have different values depending on the operating system type.

### Windows XP, 32bit:

<InstallPath>	C:\program files\Siemens\Sinumerik\VNCK\v4.5 This folder might be modified during installation.
<AllUsersPath>	C:\Documents and Settings\All Users\Application Data\Siemens\Sinumerik\VNCK\v4.5 Fixed Path
<UserPath>	C:\Documents and Settings\<UserName>\Local Settings\Application Data\Siemens\Sinumerik\VNCK\v4.5 Fixed Path
<HMIAdvPath>	C:\HMIAdv This folder might be modified during installation.
<OperatePath>	C:\Siemens\Sinumerik\HMIsv4.5.2 This folder might be modified during installation.

The standard path for applications  
(...\program files\...) must not be used!

### Windows XP, 64bit:

<InstallPath>	C:\Siemens\Sinumerik\VNCK\v4.5 This folder might be modified during installation.  The standard path for 32bit applications (...\program files (x86)\...) must not be used!.
<AllUsersPath>	C:\Documents and Settings\All Users\Application Data\Siemens\Sinumerik\VNCK\v4.5 Fixed Path
<UserPath>	C:\Documents and Settings\<UserName>\Local Settings\Application Data\Siemens\Sinumerik\VNCK\v4.5 Fixed Path
<HMIAdvPath>	C:\HMIAdv This folder might be modified during installation.
<OperatePath>	C:\Siemens\Sinumerik\HMIsv4.5.2 This folder might be modified during installation.  The standard path for 32 bit applications (...\program files (x86)\...) must not be used!

### Windows 7, 32bit

<InstallPath>	C:\program files\Siemens\Sinumerik\VNCK\v4.5 This folder might be modified during installation.
<AllUsersPath>	C:\Program Data\Siemens\Sinumerik\VNCK\v4.5 Fixed Path
<UserPath>	C:\Users\<UserName>\AppData\Local\Siemens\Sinumerik\ VNCK\v4.5 Fixed Path
<HMIAdvPath>	C:\HMIAdv This folder might be modified during installation.
<OperatePath>	C:\Siemens\Sinumerik\HMIsv4.5.2 This folder might be modified during installation.  The standard path for 32 bit applications (...\program files (x86)\...) must not be used!

## Windows 7, 64 bit

<InstallPath>	C:\Siemens\Sinumerik\VNCK\v4.5 This folder might be modified during installation.
	The standard path for 32bit applications (...\program files (x86)\...) must not be used!
<AllUsersPath>	C:\Program Data\Siemens\Sinumerik\VNCK\v4.5 Fixed Path
<UserPath>	C:\Users\<UserName>\AppData\Local\Siemens\Sinumerik\VNCK\v4.5 Fixed Path
<HMIAdvPath>	C:\HMIAdv This folder might be modified during installation.
<OperatePath>	C:\Siemens\Sinumerik\HMIsv\v4.5.2 This folder might be modified during installation.
	The standard path for 32 bit applications (...\program files (x86)\...) must not be used!

## 1.5 Silent Installation

The VNCK setup supports silent installation.  
To install VNCK in silent mode please call

setup.exe -OF:"<Path>\opfile.txt"

The opfile contains the information needed for silent installation. Please refer to the doc folder in the installation path to find an example for the opfile structure.  
The silent installation does not include an automatic reboot. This needs to be done by the calling frame setup.

---

### Note

When running a silent setup on Windows 7 operating system an additional confirmation might be necessary to begin with the installation process.

---

## 1.6 Documentation

In folder <InstallPath>\doc you will find the following documentation files:

- VNCK\_Reference.pdf (this document)  
That's the user manual describing how to use the interface to VNCK and there are information about the functions and their parameters.
- VNCK\_Restrictions.pdf  
This document describes the actual restrictions of the VNCK system.
- VNCK\_ReleaseNotes.pdf  
This document describes changes from the last VNCK version to the actual version as well as additional information about new VNCK features. The changes usually refer to new or changed interfaces or type definitions. The modifications are caused by SINUMERIK 840D kernel functionality or VNCK server behavior.
- VPLC\_IO.pdf  
This document describes the interface to the VPLC subsystem as well as a basic understanding how this virtual VPLC component is working.
- ReadMe\_OSS.pdf  
This document describes license issues due to open source software used in the VNCK software.
- ReadMeHMIAdvanced\_OSS.pdf  
This document describes license issues due to open source software used in the software package HMI Advanced.
- ReadMeHMIOperate\_OSS.pdf  
This document describes license issues due to open source software used in the software package SINUMERIK Operate.
- VNCK\_server\_versions.txt  
File describing the versions of the components of the VNCK system.
- channelStateStopcond.txt  
List of stop conditions of the NCK channel state.
- opfile.txt  
Example file for silent installation.
- theVNC.ini  
Example file for user specific theVNC.ini file.
- theVNC\_ini.txt  
Documentation of sections and entries of theVNC.ini.
- vmcp\_dll.chm
- Interface documentation for Virtual Machine Control Panel interface vmcp.dll.
- VNCKView\_Description.pdf  
Documentation of VNCKView utility.



## 1.7 Additional Documentation for Developers

During installation of VNCK additional information for software developers can be installed to folder <InstallPath>\sw.

### Implementation of VNCK COM interface

Following files describe the VNCK COM interface and can be processed by a MIDL compiler:

- idl files
- theVNC.idl
- vncDefines.idl
- vncTypes.idl

As an alternative the executable 'theVNC.exe' can be imported.

Following files describe error and warning code definitions, enums and structs used in the VNCK COM interface:

- vncDefinesBase.h
- vncEnumsBase.h
- vncTypesBase.h

---

#### Note

The file "vncDefines.idl" describes all error codes generated by VNCKServer and HMI Services used in VNCK and the NCK kernel.

---

### Implementation of VPLC interface:

Following files describe the standard C interface to a shared memory provided by VPLC containing the I/O data VPLC uses to communicate with simulated peripheral devices:

- vplc.h
- vplc\_def.h
- vplc\_io.h

The required lib/dll files are:

- iVPLC.lib
- iVPLC.dll

Implementation of a virtual machine control panel (vMCP):

- vmcp\_int.h The file describes the interface to create a virtual machine control panel.
- vmcp.dll Is the required dll file.



## 2 System Characteristics

### 2.1 Chronological Sequence of Program Processing

All events arising from the processing of a part program inside the VNCK have a common parameter 'Virtual Time Stamp'. This 'stamp' describes the virtual real time when the corresponding data or information are generated or become active. Since the VNCK is built up from the same software modules as a real target machine executable program, the VNCK internal clock is implemented in the same way. There are several counters inside the VNCK. In combination with the machine data of the IPO cycle time it is possible to compute the above-mentioned time stamp as a value representing an original real clock value. Depending especially on the parameterizing of the program process control, in most cases the VNCK will work much faster than a real machine. But in some cases, for example when each fine IPO cycle reports actual values to the simulator, it may take more time to run a program. But in all cases the simulator can assign the VNCK-produced data and information via the time stamp to a time in real life of program processing.

## 2.2 Asynchronous Communication Simulation <-> VNC Server

Generally communication between the simulator and the VNCK system is handled asynchronously in both directions. This means no application has to wait for the result of a command inside the function call. Commands to the server (VNCFunction (...)) are taken over by the server and, after a quick parameter check, are returned immediately. After a further check the VNC server works on the command eventually using further HMI servers and / or the VNCK kernel. This work is finished in calling the simulation interface (SIMFctResponse (...)). In the same way the simulator has to work on the data the VNCK system is reporting.

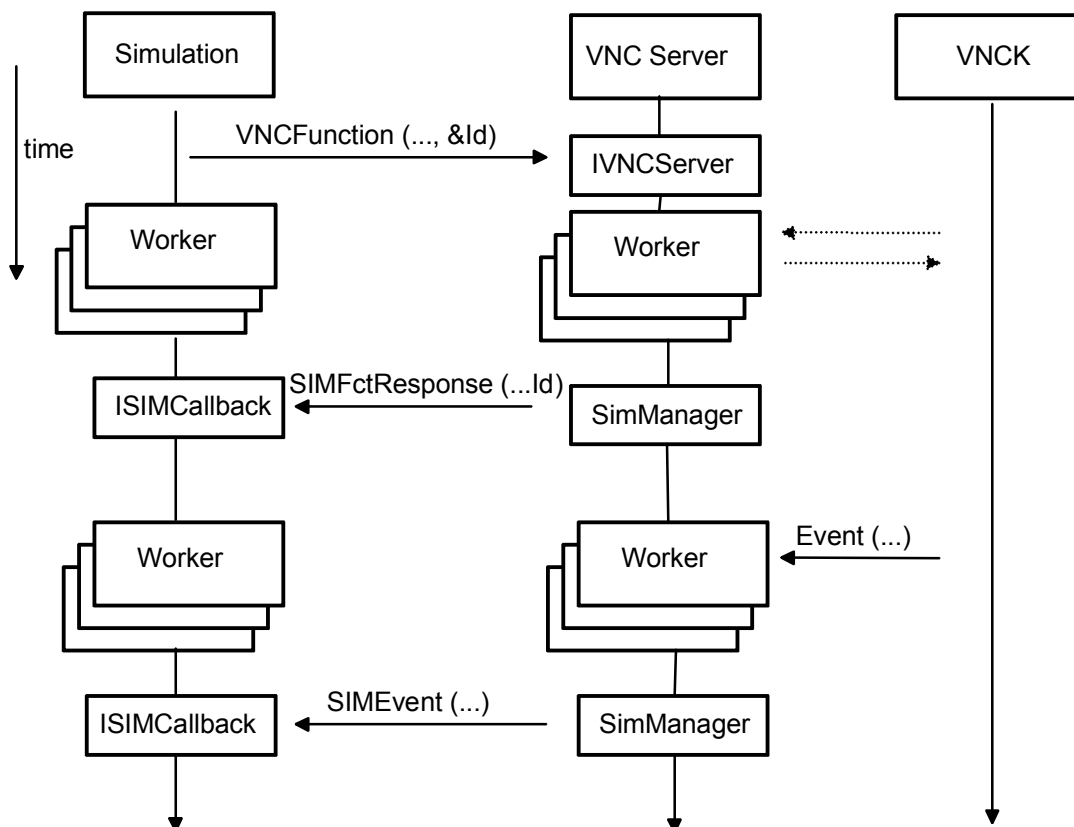


Figure 2-1: Asynchronous Communication Simulation <-> VNC Server

Nearly all VNCFunction(...) calls return an identifier by setting &Id. This identifier can be used to assign simulation callbacks to previously sent VNCK commands. In very few cases the VNCK server works synchronously. Then a special warning code is return by VNCFunction(...) to notify the simulation that no response callback will be sent.

## 2.3 Interfaces

The next chapters describe groups of interfaces that are used to handle different use cases or VNCK facilities that can be used to simulate the real world as well as possible.

Nowadays it is possible to combine ISV license handling as well with a single NCU processing environment as with NCU link systems.

VPLC processing can actually only be used with a single NCU processing environment.

### 2.3.1 Single NCU Processing

**IVNCServer / ISIMCallback, ISIMCallback\_ext, ISIMCallback\_sa**

These interfaces are used by simulation when there is a single NCU to be controlled by the VNCK system.

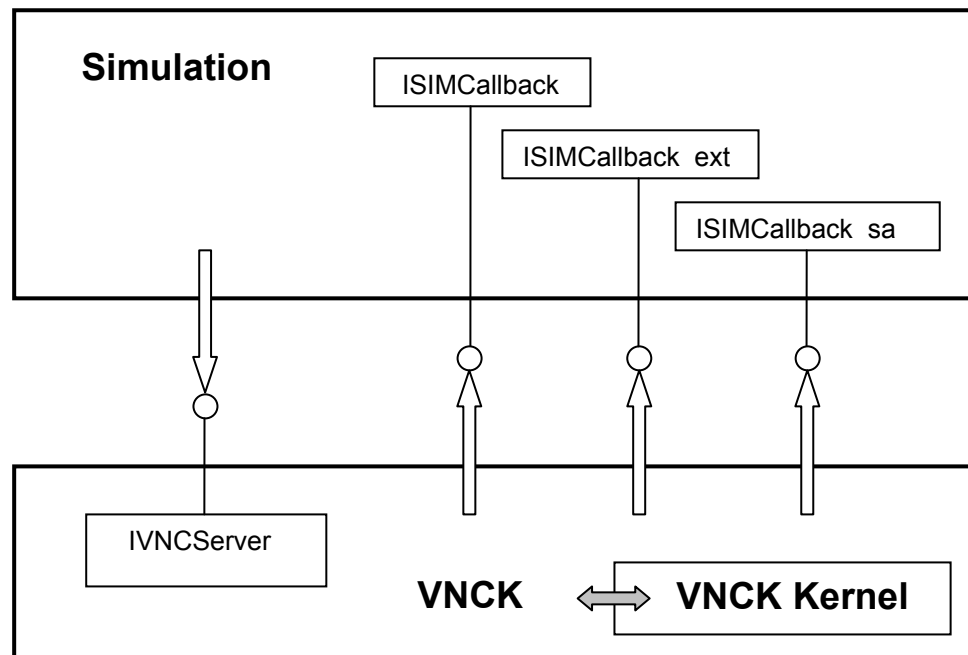


Figure 2-2: Single NCU Processing

### 2.3.2 VPLC Processing

#### IVPLC / ISIMVPLCCallback / iVPLC.dll, vmcp.dll

The interfaces IVPLC and ISIMVPLCCallback allow simulation to access and control a VPLC instance inside the VNCK system. In addition to the COM interfaces there are additional dll files. 'iVPLC.dll' enables simulation to synchronize simulation and VPLC directly in view of a common data IO interface memory. 'vmcp.dll' offers methods for implementing a virtual control panel.

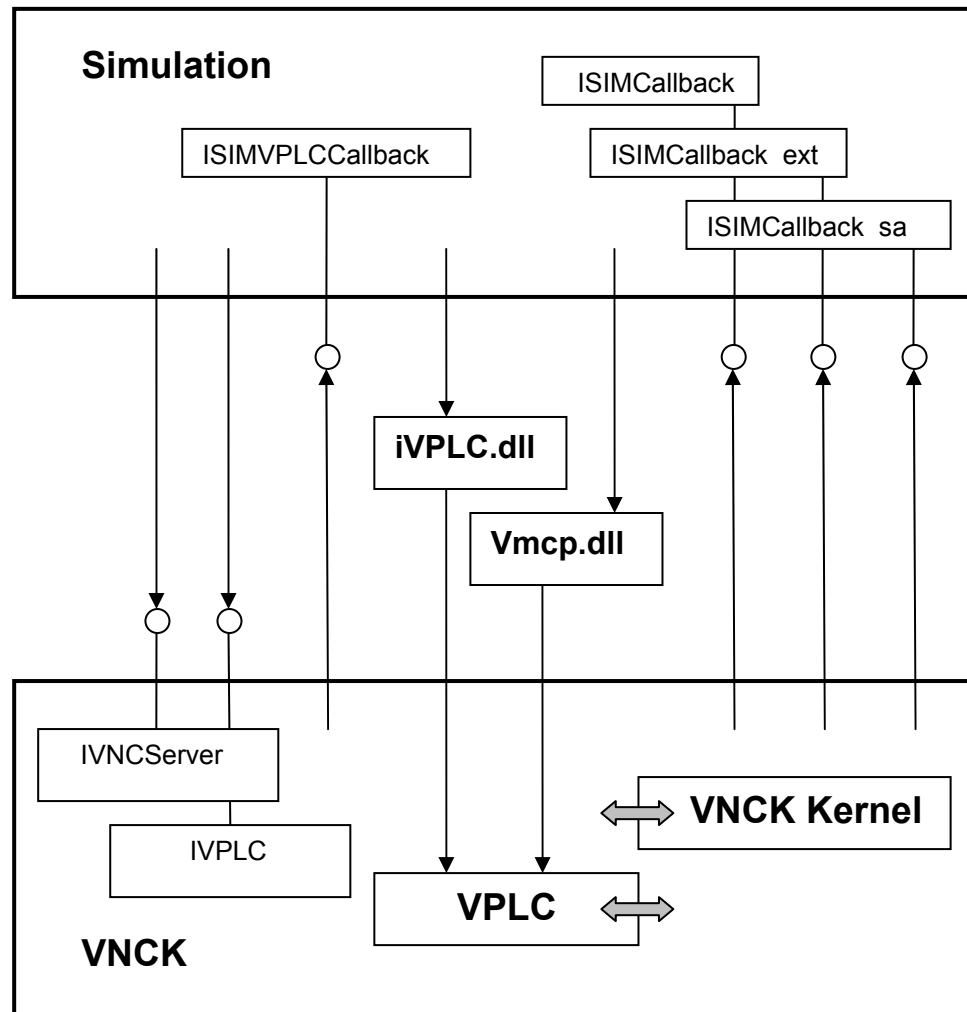


Figure 2-3: VPLC Processing

### 2.3.3 NCU Link Processing

#### IVNCLinkServer / ISIMLinkCallback

#### IVNCNcuServer / ISIMNcuCallback, ISimLinkNcuFactory

The interfaces IVNCLinkServer and the according ISIMLinkCallback are used to handle link units as control instances of the NCU units that are synchronized via ncuLink. The interfaces IVNCNcuServer and ISIMNcuCallback are used for the extended and changed functions of NCU units of a link system instead of single NCUs running without ncuLink synchronization. For simulation comfort startup of a link system another callback interface ISimLinkNcuFactory is necessary. The following picture shows the possible configuration of a link unit with two NCU units.

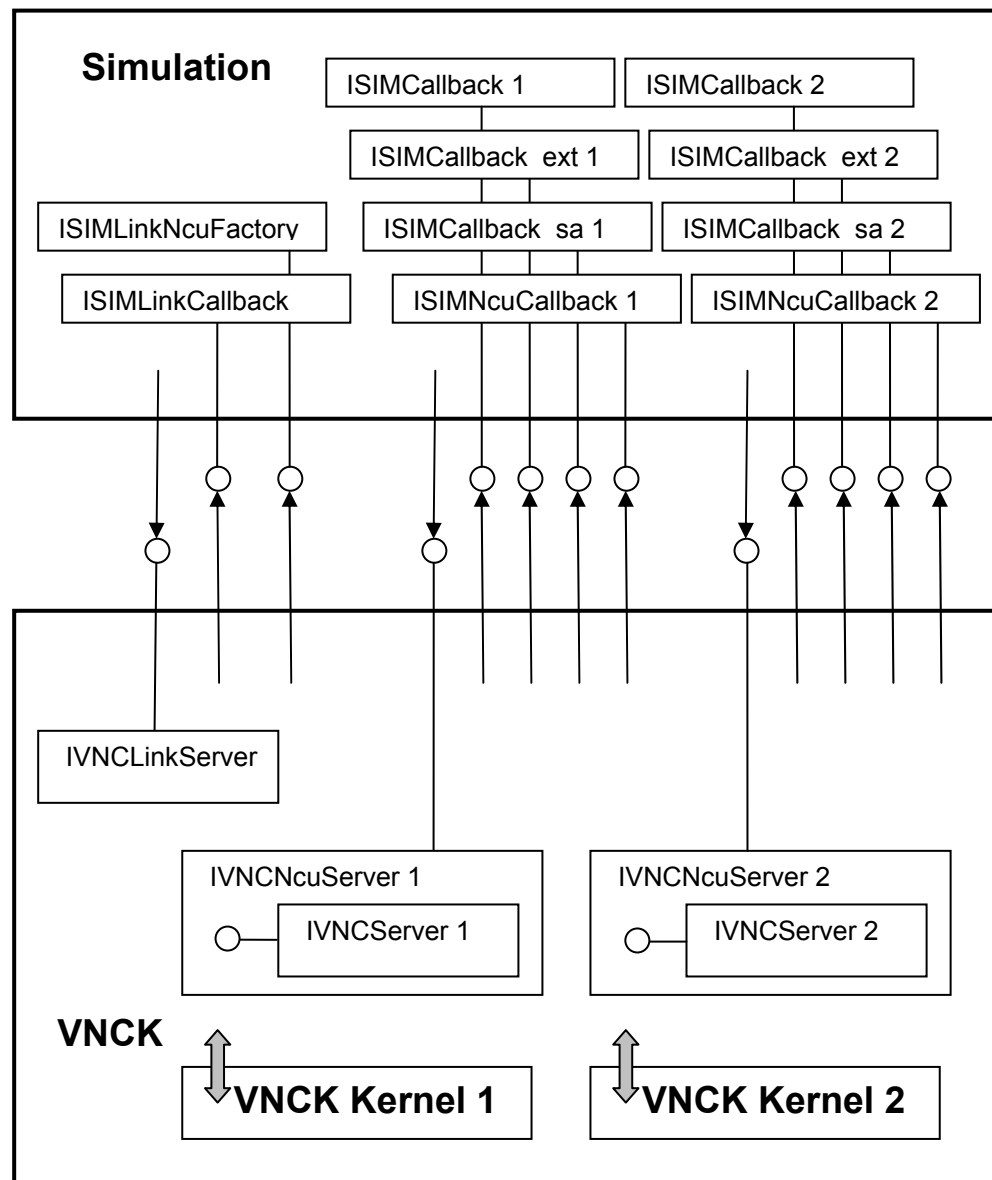


Figure 2-4: NCU Link Processing

### 2.3.4 License Handling

#### ISIMCallbackLicense

This interface will be used to fulfill license requirements with ISV customers of the VNCK system.

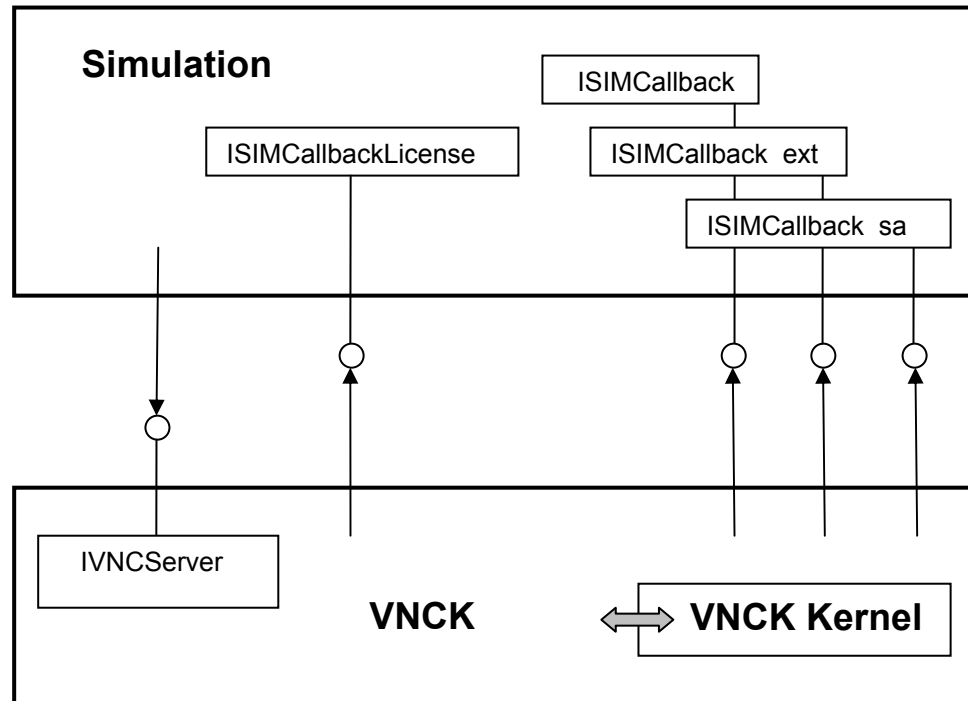


Figure 2-5: License Handling



## 2.4 Interface Function Return Values

If no error has occurred performing a command or request function of an interface to the VNCK system the function returns a value 0 ( zero ). Otherwise a return code described in the file 'vncDefinesBase.h' is returned. This file resides in the directory '.../vnck/sw'.

---

### Note

Several of the interface functions return a positive value not zero to advise simulation to either a function specific behaviour or to the fact that a usually corresponding callback function will not be issued since this function works synchronously.

---

## 2.5 Freeze Handling in Simulation

The communication between VNCKServer and an external simulation application during simulation runtime is based on a combination of VNCKRun and SIMFreeze calls. One base rule when implementing the VNCK COM Interface is that each SIMFreeze call sent to simulation must be answered immediately with a VNCKRun.

The same applies for working with the link interface. Each call of SIMLinkFreeze must be answered with VNCKLinkRun.

## 2.6 Creating high performance traces

The trace mechanism of VNCK was re-engineered in version VNCK4.5. Different to older versions, trace information is now stored in a temporary shared memory file. A separate process started by ATraceGui.exe (found in VNCK installation folder) reads the shared memory content and writes it to a log file.

The performance of the new mechanism enables users to create traces parallel to simulation. Problems and failures in VNCK, which were hard to track in the past, now can be traced more easily.

The new features needs to be enabled as follows:

- Entries in theVNC.ini  
[GLOBAL]  
trace\_target=1
- Start ATraceGui.exe
- Click "start Writer" in ATraceGui.exe
- Minimize ATraceGui.exe

The existing setting trace\_server in theVNC.ini is still valid and defines, which data is sent to the trace mechanism.

The GUI of ATraceGui.exe provides some additional settings to define:

- How many trace files are to be created as ring-buffer.
- How much data is stored to one trace file.
- In which time frames the shared memory content is transferred to the trace files.

## 3 Interface Functional Description

The following chapters describe the API functions related to the VNCK server as well as the event functions of the callback interfaces.

VNCK server functions are marked by a prefix 'VNC' (e.g. VNCBoot(...)). Callback functions are marked by a prefix 'SIM' (e.g. SIMBootResponse(...)).

Asynchronous function calls to the VNCK system return “[out] long \* pActionId” as their last parameter, which represents an identifier to any callback function returned by VNCK server.

### 3.1 VNCK Boot and Shutdown

#### 3.1.1 General Information

As the VNCK is built from the identical software as a SINUMERIK 840D sl controller it can be configured in the same way as a real machine. Thus original machine data sets as well as setting data sets, definition files and cycles from a user machine can be used. There are different modes of booting the VNCK with those data files.

#### 3.1.2 Establishing the connection between controller and simulation

##### Synchronous service

```
VNCSetSIMInterface
(
    ISIMCallback*           pSim,
    ISIMCallback_ext*       pSim_ext,
    ISIMCallback_sa*        pSim_sa,
    ISIMCallbackLicense*    pSimLicense
);
```

##### Parameters

pSim	basic callback interface
pSim_ext	extended callback interface
pSim_sa	callback interface based on safeArray definition
pSimLicense	callback interface for license management

## Description

VNCSetSIMInterface() must be called to initialize the instantiate VNCK server and to establish the COM connection between VNCK server and simulation.

Simulation must send valid pointers to simulation internal callback functions for

- Basic callback functions like e.g. SIMBootResponse
- Extended callback functions like e.g. SIMRegisterCommandResponse
- SafeArray based callback functions, which are used mainly in VisualBasic and C# to return list based callbacks based on safeArray definition
- License callback functions when using the ISV license mechanism

---

### Note

VNCSetSIMInterface is a synchronous function call to VNCK server. It returns a positive value to sign a successful connection to VNCK server.

---

## 3.1.3 Controller start-up

### Asynchronous service

```
VNCBoot
(
  BSTR                      sNcuName,
  VNCBootType_t            tBootType,
  BSTR                      sBootDataPath,
  long *                   plActionId
);
```

### Event

```
SIMBootResponse
(
  VNCResult_t*              ptResult,
  long                      lActionId
);
```

### Parameters

sNcuName	name of the VNCK kernel NCU to handle
tBootType	type of initialization data
sBootDataPath	path to initialization data
ptResult	result of the boot call
[p]lActionId	identifier to callback functions

### Description

VNCBoot starts the VNCK boot sequence. Depending on the type (VNCBootType\_t) of initialization data the VNCK will initiate its boot sequence based on

- an archivefile coming from a real machine tool (VNC\_BOOTTYPE\_IBN)
- an previously generated SRAM file (VNC\_BOOTTYPE\_SRAM)
- a minimal machine configuration (VNC\_BOOTTYPE\_SIM\_DATA)

#### VNC\_BOOTTYPE\_IBN:

VNCK will boot based on an archive file retrieved from a real machine tool. Like with the real machine, this will take some time due to intermediate reboot actions depending on the complexity of the archive file.

The location of the archive file is given by parameter `sBootDataPath`.

#### VNC\_BOOTTYPE\_IBN\_CC:

Same behaviour like with boot type `VNC_BOOTTYPE_IBN`. Additionally OEM compile cycles are added to the boot sequence. These compile cycles must be stored to a folder named `"cc.dir"` parallel to the archive file and need to be named with extension `".elf"`.

#### VNC\_BOOTTYPE\_SIM\_DATA:

VNCK will boot based on a minimal configuration. Additional boot files can be stored in a boot folder given by `sBootDataPath`. These files can be initialization files (`"*.ini"`), definition files (`"*.def"`) and cycles (`"*.spf"`).

An `"initial.ini"` file the basic configuration of the machine data set.

A `"to_ini.ini"` file is used to configure all dynamic datalike tool data, R parameters, zero offsets and user data defined by GUD variables and system parameters.

`"*.spf"` files hold subprogram or cycle G code and are downloaded during the boot sequence to the VNCK internal Siemens cycle directory `"cst.dir"`.

`"*.gud"` and `"*.mac"` files hold variable and macro definitions and are downloaded during the boot sequence to the VNCK internal Siemens definition directory `"def.dir"`.

#### VNC\_BOOTTYPE\_SIM\_DATA\_CC:

Same behaviour like with boot type `VNC_BOOTTYPE_SIM_DATA`. Additionally OEM compile cycles are added to the boot sequence. These compile cycles must be stored to a folder named `"cc.dir"`, which is a subfolder in the boot folder and need to be named with extension `".elf"`.

Once the boot sequence based on archive files or minimal configuration is successfully finalized, it is possible to store a so-called SRAM file, which contains a binary representation of the actual controller state of VNCK. Further on these SRAM files can be used to boot the VNCK system, which is much faster compared to booting based on archive files.

The typical usecase for simulation applications is to boot only from SRAM files. Booting from archive or based on minimal configuration basically represents the authoring step to get a valid SRAM file.

#### VNC\_BOOTTYPE\_SRAM:

VNCK will boot based on a binary representation of the VNCK controller state, the so-called SRAM file. The path to the SRAM file is given with parameter `sBootDataPath`.

#### VNC\_BOOTTYPE\_SRAM\_CC:

Same behaviour like with boot type `VNC_BOOTTYPE_SRAM`. Additionally OEM compile cycles are added to the boot sequence. These compile cycles must be stored to a folder named `"cc.dir"` parallel to the SRAM file and need to be named with extension `".elf"`.

#### VNC\_BOOTTYPE\_SRAM\_SAVE:

Same behaviour like with boot type `VNC_BOOTTYPE_SRAM`. Once the API function `VNCShutdown` is called, the actual controller state will be stored to the SRAM file, which originally was used to boot VNCK.

The parameter sNcuName is internally used to communicate with HMI services. When working with VNCK stand-alone the name VNCK is used. When working with VNCK plus VPLC the name VNCKVPLC is used.

Once the boot sequence is finalized the event SIMBootResponse is issued by VNCK server. Before SIMBootResponse is sent additional events will be generated by VNCK server. This are basically:

- SIMVNCCConfig giving basic information on the NCK system
- SIMChannelConfigChanged sending axis information for each channel of the running VNCK system (depending on the complexity of archive/SRAM).

In VNCK4.5 additional timeout values were introduced to the Global section of theVNC.ini:

- timeout\_all\_booting\_sram defines the maximum time between VNCBoot and SIMBootResponse, when booting with boottype VNC\_BOOTTYPE\_SIM\_DATA and VNC\_BOOTTYPE\_SRAM
- timeout\_all\_booting\_arc defines the maximum time between VNCBoot and SIMBootResponse, when booting with boottype VNC\_BOOTTYPE\_IBN

---

#### Note

Since an SRAM file is a binary representation of the VNCK controller state, the SRAM file must only be used with the same version of VNCK it was generated with.

VNCK will check the version information of an SRAM file and sends error VRV\_BOOT\_WRONG\_VNCK\_VERSION, if versions are not matching. Also the NCK software will check whether the SRAM file was created with the same version. It will create an NCK alarm message, if it doesn't match.

To be able to use boot type VNC\_BOOTTYPE\_IBN a SINUMERIK 840D sl archive file is needed. It must only contain NC archive information. Additional archive data for PLC, drives or user data will cause errors.

---

#### Note

*SIMBootResponse* will send a value VRV\_PROGRAM\_STARTED\_AND\_RUNNING\_INSIDE\_BOOTING, if NC programs or cycles were started during boot phase, but haven't finished until the end of the boot phase.

---

#### Note

If ShopMill or ShopTurn is used on the particular Machine a special compile cycle *machgen.elf* must be loaded to ensure back and forth compilation of JobShop programs. The compile cycle is delivered in folder:

<InstallPath>\ncRoot\siemens\sinumerik\cycles\loa

To boot VNCK including this compile cycle the according xx\_CC boot options must be used.

---

---

#### Note

When preparing an NC archive to run with VNCK it must be assured that the following MDs show the same value. This applies for all axes AXn in the archive file.

N30110 \$MA\_CTRLOUT\_MODULE\_NR[0,AXn]  
N30220 \$MA\_ENC\_MODULE\_NR[0,AXn]  
N30220 \$MA\_ENC\_MODULE\_NR[1,AXn]

---

### 3.1.4 Controller Status Saving

#### Asynchronous service

```
VNCSaveData
(
  VNCSaveData_t          tSaveMask,
  BSTR                    sFileName,
  long *                  plActionId
);
```

#### Event

```
SIMSaveDataResponse
(
  VNCRResult_t*          ptResult,
  long                    lActionId
);
```

#### Parameter

ptResult	Resulting value of function
tSaveMask	identification of data to save
sFileName	name of destination file
[p]lActionId	Identifier to callback functions

#### Description

Depending on the type of booting by a SRAM file or a startup archive or independent of this way the actual state of the complete VNCK or single parts of data of the simulation machine will be saved for future startups of the VNCK. The parameter tSaveMask described which data of the actual running VNCK kernel has to be stored.

The function returns some particular error codes, if the given sFileName is invalid or empty.

VNC\_SAVEDATA\_SRAM:

Using this value either the complete state of the last shutted down VNCK kernel or the state of the actual running VNCK kernel is stored to a file 'vmfSim.dat'. This is a 'zip' file that contains several file representing the NCU state. You can use this file to a further VNCBoot() call with a boot mode VNC\_BOOTBASICTYPE\_SRAM.

#### VNC\_SAVEDATA\_IBN:

Using this value the VNCK server asks the HMI archive server to build a startup archive file from the running VNCK kernel machine. This file is stored in the VNCK directory as 'archiveSim.arc'. You can use this file to a further VNCBoot() call with a boot mode VNC\_BOOTBASICTYPE\_IBN.

In VNCK4.5 additional timeout values were introduced to the Global section of theVNC.ini:

- timeout\_all\_saving\_sram defines the maximum time between VNCSaveData and SIMSaveDataResponse, when storing with type VNC\_SAVEDATA\_SRAM
- timeout\_all\_saving\_arc defines the maximum time between VNCSaveData and SIMSaveDataResponse, when storing with type VNC\_SAVEDATA\_IBN.

---

#### Note

In the case of using VNC\_SAVEDATA\_SRAM after shutting down the NCU kernel this function is implemented as a synchronous function. Therefore no actionId is returned and the response event is not fired. There is a warning value notifying the synchronous processing.

An event SIMKernelResetEvent() with a parameter VNC\_KERNEL\_RESET\_EVENT\_SAVEDATA\_SRAM is fired to notify simulation about a kernel reset occurring processing the request.

---

#### Note

VNCSaveData will be ignored, if slice mode VNC\_SLICEMODE\_NOT\_INCYCLE is active. The error SYSTEM\_ACTIVE\_WITH\_NOT\_INCYCLE will be sent in this case. Before using VNCSaveData VNC\_SLICEMODE\_NOT\_INCYCLE must be deactivated manually.

---

#### Note

Creation of archive files may take longer than ten minutes depending on the system resources of your computer.

---

#### Note

Make sure that there is enough disk space to save the data. It is recommended to have at least 3 times the size of an SRAM file available on the disk. This applies for the C drive as well as for the target drive. The C drive is used to temporarily store the data before moving it to the target drive.

---



### 3.1.5 Controller Status Refreshing

#### Asynchronous service

```
VNCMatchData
(
  BSTR                                sMachineName,      = NULL; not yet used
  VNCMatchData_t                     tRefreshMask,
  long *                              plActionId
);
```

#### Event

```
SIMMatchDataResponse
(
  VNCResult_t*                       ptResult,
  long                               lActionId
);
```

#### Parameter

ptResult	Resulting value of function
sMachineName	name of the VNCK machine to handle
tRefreshMask	identification of data to refresh
[p]lActionId	Identifier to callback functions

#### Description

Using VNCMatchData the VNCK will be updated by the dates and values of the described machine. Depending on the mask different dates, i.e. machine data, tool data, guides, cycles, ..., types will be attached

VNC\_MATCHDATA\_SRAM:

Using this value the complete state of the running VNCK is updated from a stored SRAM file 'vmfSim.dat'. This method prevents simulation from shutting down and rebooting a VNCK system to restore a VNCK state.

---

#### Note

Until now only the value VNC\_MATCHDATA\_SRAM can be used. Furthermore the NCU had to be booted with a bootType VNC\_BOOTBASICTYPE\_SRAM.

An event SIMKernelResetEvent() with a parameter VNC\_KERNEL\_RESET\_EVENT\_MATCHDATA\_SRAM is fired to notify simulation about a kernel reset occurring processing the request.

---



---

#### Note

VNCMatchData will be ignored, if slice mode VNC\_SLICEMODE\_NOT\_INCYCLE is active. The error SYSTEM\_ACTIVE\_WITH\_NOT\_INCYCLE will be sent in this case. Before using VNCMatchData VNC\_SLICEMODE\_NOT\_INCYCLE must be deactivated manually.

---

### 3.1.6 Resetting the VNCK Kernel

#### Asynchronous service

```
VNCResetKernel  
(  
    long *                                plActionId  
);
```

#### Event

```
SIMResetKernelResponse  
(  
    VNCRResult_t*                        ptResult,  
    long                                lActionId  
);  
  
SIMKernelResetEvent  
(  
    VNCKernelResetEventReason_t tReason  
);
```

#### Parameter

ptResult	Result of resetting the VNCK kernel
tReason	Reason for resetting
[p]lActionId	Identifier to callback function

#### Description

Calling VNCResetKernel() causes a 'warmboot' of the VNCK kernel. Thus alarms requiring this action can be deleted or downloaded data requiring this action can become active. The event SIMKernelResetEvent is always fired whenever the VNCK server detects the kernel has (re-) booted on simulations request. There are several other reasons than VNCResetKernel() that can cause the event. The value of tReason reports the reason for SIMKernelResetEvent().

---

#### Note

VNCResetKernel behaviour is different to VNCK2.1 and older. Between VNCResetKernel and SIMResetKernelResponse SIMFreeze events may appear, which must be answered with VNCRun.

---

### 3.1.7 Controller Shutdown

#### Asynchronous service

```
VNCShutdown  
(  
    long *                plActionId  
);
```

#### Event

```
SIMShutdownResponse  
(  
    VNCResult_t*          ptResult,  
    long                  lActionId  
);
```

#### Parameter

ptResult	Resulting value of shut down
[p]lActionId	Identifier to callback functions

#### Description

Shuts down the virtual controller. The VNCK Server will continue to exist after the shutdown. By calling VNCBoot a new VNCK can be booted.

In VNCK4.5 an additional timeout value was introduced to the Global section of theVNC.ini:

- timeout\_all\_shutting defines the maximum time between VNCShutdown and SIMShutdownResponse, when storing with type VNC\_SAVEDATA\_SRAM.

---

#### Note

After calling VNCShutdown the VNC COM interface must be re-initialized by calling VNCSetSIMInterface before calling the next VNCBoot.

---

### 3.1.8 Initializing the Channel Axes Values

#### Asynchronous service

```
VNCSetInitialChanAxesValues
(
    long                    IChannel,
    VNCCoordSys_t          tCoordSys,
    long                    INumber,
    long *                  plChanAxIndex,
    double *                pdInitialChanAxValue,
    long *                  plActionId
);
```

#### Event

```
SIMSetInitialChanAxesValuesResponse
(
    VNCRResult_t*          ptResult,
    long                    lActionId
);
```

#### Parameter

ptResult	Resulting value of function
tCoordSys	Type of axes values coordinate system
INumber	Number of following axes values
plChanAxIndex	Array of channel axes indexes
pdInitialChanAxValue	Array of initial axes values
[p]lActionId	Identifier to callback functions

#### Description

Using this function the simulation is able to initiate the channel axes to matching values inside all limits of restrictions. This feature can be seen as a substitute of a reading of actual axes positions.

---

#### Note

There is another version of this function:  
VNCSetInitialChanAxesValuesSa ( ... );

---

#### Note

VNCSetInitialChanAxesValues will send a value  
VRV\_AXIS\_NOT\_REFERENCED, if it is called and at least one channel axis has not reached its reference position. This signs that setting the axis value for this axis will probably fail.

---

### 3.1.9 Setting VNCK System Sleeptime

#### Synchronous service

```
VNCSetLtcSleepTime  
(  
    long                                ISleepTime  
);
```

#### Parameter

ISleepTime	SleepTime parameter
------------	---------------------

#### Description

Using this function it is possible to activate a VNCK process sleep. Thus other applications are able to use processor idle time. Nevertheless the VNCK processes are not parameterized with a higher process priority or class. They only use the idle time that windows OS distributes. The sleep call inside the VNCK system is activated whenever the kernel process notifies itself to the server when no other events are sent, f. e. those events describing a part program being processed. This watchdog function is parameterized by the LIFETIMECHECK entry of [VNCK] section in the VNC.ini file.

---

#### Note

In fact, this function is implemented as a synchronous function. Therefore a warning value is returned to the request to advise to this behavior.

---

### 3.1.10 Setting the VNCK Kernel Process State

#### Synchronous service

```
VNCSetKernelProcessState  
(  
    VNCKernelProcessStateType_t    tKernelProcessState  
);
```

#### Parameter

tKernelProcessState	Process state active / idle
---------------------	-----------------------------

#### Description

Using this function the VNCK kernel process can be idled. If no part program is processed and no other commands have to be executed on the kernel process it may be helpful to idle the VNCK kernel process to spend all the time this process - it always runs - to other applications.

---

#### Note

In fact, this function is implemented as a synchronous function. Therefore a warning value is returned to the request to advise to this behavior.

---

## 3.2 VNCK Configuration services

### 3.2.1 General Information

Within the boot sequence the VNCK will send events to the simulator transmitting the VNCK configuration with regard to the general kernel configuration as well as to all channel settings.

### 3.2.2 Retrieving the NC configuration

#### Asynchronous service

```
VNCGetVNCKConfig
(
    long *                                pActionId
);
```

#### Event

```
SIMVNCKConfig
(
    VNCKResult_t*                        ptResult,
    VNCKVersion_t*                       ptVersionBooted,
    long                                INumChannels,
    double                              dIpoCycleTime,
    long                                IActionId
);
```

#### Parameters

ptResult	Resulting value
ptVersionBooted	booted VNCK version
INumChannels	the current number of channels
dIpoCycleTime	VNCK IPO cycle time in milliseconds
[p]IActionId	Identifier to callback functions

#### Description

Before responding to the VNCBoot order the VNCK will inform the simulator about several VNCK basic data via the event SIMVNCKConfig. The action identifier should refer to the boot function action identifier. The ptVersionBooted parameter describes the version of the original SINUMERIK 840D sl Controller.

Calling VNCGetVNCKConfig() will respond with the event SIMVNCKConfig, too.

### 3.2.3 Retrieving the axes configuration

#### Asynchronous service

```
VNCGetChannelConfig  
(  
    long          IChannel,  
    long *        pActionId  
);
```

#### Event

```
SIMChannelConfigChanged  
(  
    VNCResult_t*          ptResult,  
    double                dVirtTime,  
    long                  IChannel,  
    BSTR                  sChannelName,  
    long                  INumber,  
    VNCAxisConfDataArray_t* ptAxisConfData,  
    long                  IActionId  
);
```

#### Parameters

ptResult	Resulting value of shut down
tVirtTime	Time stamp
IChannel	the required or current channel
sChannelName	the logical name of the channel
INumber	the number of channel axes data sets
ptAxisConfData	array of channel axes data sets
IActionId	Identifier of the event function

#### Description

With this call, the simulator determines the kinematic configuration of the VNCK during the booting process at first and during the program execution every time the configuration changes (for example if any axes leave or change the channel). This tells the simulator how many axes have been configured, as well as the corresponding axis types, the values of the software limit switches, and the names of the individual axes. The simulator can then use this information to assign the VNCK axes to the axes of its internal kinematic model. This event is sent during the boot sequence for each channel.

Calling VNCGetChannelConfig () will result in the same response events as sent while booting the NCU.

---

#### Note

There is another version of the event:  
SIMChannelConfigChangedSa ( ... );

---



### 3.2.4 Retrieving the VNCK Server Version

#### Synchronous service

```
VNCGetServerVersion  
(  
  VNCVersion_t *          ptVersionServer,  
  long *                  plDevNr  
);
```

#### Parameters

ptVersionServer	Server version
plDevNr	VNCK development number

#### Description

The function can be called at any time to retrieve the version of the VNCK server.

---

#### Note

In fact, this function is implemented as a synchronous function. Therefore a warning value is returned to the request to advise to this behavior.

---

## 3.3 Domain Data Management

### 3.3.1 General Information

Depending on the history the term 'project' will be used to describe the place where part programs and data belonging to the machining of a work piece are stored. In an 840D sl HMI context the term 'work piece directory' is used. Internally the 'projects' are represented by 'work piece directories'. The term 'file' is often used to describe all the data objects representing part programs or nc data.

#### Description of variable transfer info mode

If files are transferred between the VNCK Server on the simulation side and VNCK as the machine, it is possible to set the mode of information. The simulator can order one event describing the result of the complete transfer of all files or of each single file at its end. Otherwise the simulator will receive several events describing the actual state of each single transfer described by an perceptual value.

```
typedef enum      {          VNC_FILETRANSFER_ALL_CLOSED,  
                   VNC_FILETRANSFER_SINGLE_CLOSED,  
                   VNC_FILETRANSFER_OPEN          }  
VNCTransInfoMode_t;
```

### 3.3.2 Retrieving existing project directories

#### Asynchronous service

```
VNCGetProjectList  
(  
    long *                                plActionId,  
);
```

#### Event

```
SIMGetProjectListResponse  
(  
    VNCResult_t*                        ptResult,  
    long                                INumber,  
    BSTR *                              psProjects,  
    long                                lActionId  
);
```

#### Parameters

ptResult	Resulting value
INumber	Number of project names
psProjects	Array of path names of exist. project directories
[p]lActionId	Identifier to callback functions

#### Description

Retrieves the list of project directories that exist in a project management directory. These are all the work piece directories.

---

##### Note

There is another version of this event:  
SIMGetProjectListResponseSa( ... );

---

### 3.3.3 Retrieving a program list within a project directory

#### Asynchronous service

```
VNCGetProgramList  
(  
  BSTR                                sNcProjectName,  
  long *                             plActionId  
);
```

#### Event

```
SIMGetProgramListResponse  
(  
  VNCResult_t*                       ptResult,  
  long                                INumber,  
  BSTR *                             psProgramNames,  
  long *                             psProgramSizes,  
  BSTR *                             psProgramDates,  
  long                                lActionId  
);
```

#### Parameters

ptResult	Resulting value
sNcProjectName	Name of project directory
INumber	Number of program names
psProgramNames	Array of names of existing programs
psProgramSizes	Array of sizes of existing programs
psProgramDates	Array of date information of existing programs
[p]lActionId	Identifier to callback functions

#### Description

Retrieves the list of files / programs that exist in a project directory.

---

#### Note

There is another version of this event:  
SIMGetProgramListResponseSa ( ... );

---

### 3.3.4 Transferring a project to the VNCK

#### Asynchronous service

```
VNCPutProject
(
  BSTR          sProjectPathSource,
  BSTR          sNcProjectNameDest,
  VNCTransInfoMode_t tTransInfoMode,
  long *        plActionId
);
```

#### Event

```
SIMPutProjectResponse
(
  VNCResult_t* ptResult,
  BSTR          sNcProjectPath
  long          lActionId
);
```

#### Parameters

ptResult	Resulting value
sProjectPathSource	File path of source project
sNcProjectNameDest	File path of destination project
tTransInfoMode	Mode of information
sNcProjectPath	VNCK internal project path
[p]lActionId	Identifier to callback functions

#### Description

Transfers a project directory to the VNCK project management directory. If the name of the destination project is not specified (String of length 0), either a default name will be used or the work piece name from the source path is used. Otherwise the destination name must have the extension '.wpd'. Any existing project of the same name will be overwritten.

After checking the parameters and the actual status of VNCK the response event will either report success or failure of a closed transfer or the event will deliver an ID identifying the data transfer. SIMTransferFileStatus() will send events for the status and finishing of transfer.

### 3.3.5 Transferring a program to a project directory

#### Asynchronous service

```
VNCPutProgram
(
  BSTR          sProgramPathSource,
  BSTR          sNcProgramPathDest,
  VNCTransInfoMode_t tTransInfoMode,
  long *        plActionId
);
```

#### Event

```
SIMPutProgramResponse
(
  VNCTransInfoMode_t* ptResult,
  BSTR          sNcProgramPath
  long          lActionId
);
```

#### Parameters

ptResult	Resulting value
sProgramPathSource	File path and name of source file
sNcProgramPathDest	Destination directory
tTransInfoMode	Mode of information
sNcProgramPath	VNCK internal program path
[p]lActionId	Identifier to callback functions

#### Description

Transfers a program / file to a project directory. If the name of the destination does not refer to a work piece directory a default work piece name will be used. If the destination doesn't describe a program name the name of the source program will be used to store the program by its source name. Any existing program / file of the same name will be overwritten. After checking the parameters and the actual status of VNCK the response event will either report success or failure of a closed transfer or the event will deliver an ID identifying the data transfer. SIMTransferFileStatus() will send events for the status and finishing of transfer if the tTransInfoMode is appropriately set.

### 3.3.6 Transferring a setting data file to VNCK

#### Asynchronous service

```
VNCPutIniData
(
  BSTR                               sIniDataSource,
  VNCTransInfoMode_t                tTransInfoMode,
  long *                             plActionId
);
```

#### Event

```
SIMPutIniDataResponse
(
  VNCTransInfoMode_t                tTransInfoMode,
  long *                             plActionId,
  long *                             plResult
);
```

#### Parameters

plResult	Resulting value
sIniDataSource	Path of ini data file
tTransInfoMode	Mode of information
[p]lActionId	Identifier to callback functions

#### Description

Transfers an ini data file to the VNCK machine. If the name of the destination must refer to a correct setting data file, no destination is required because this file is interpreted immediately during loading and the described values are set. After checking the parameters and the actual status of VNCK the response event will either report success or failure of a closed transfer or the event will deliver an ID identifying the data transfer. SIMTransferFileStatus() will send events for the status and finishing of transfer if the tTransInfoMode is appropriately set.

### 3.3.7 Copying a program from VNCK to a local folder

#### Asynchronous service

```
VNCGetProgram
(
  BSTR                                sNcProgramPathSource,
  BSTR                                sProgramPathDest,
  VNCTransInfoMode_t                 tTransInfoMode,
  long *                              plActionId
);
```

#### Event

```
SIMGetProgramResponse (
  VNCTransInfoMode_t*   ptResult,
  long                   lActionId
);
```

#### Parameters

ptResult	Resulting value
sNcProgramPathSource	Program path to copy to local folder
sProgramPathDest	Program path to local folder
tTransInfoMode	Mode of information
[p]lActionId	Identifier to callback functions

#### Description

Copies a program from VNCK to a local folder. If sNcProgramPathSource contains only a program name without further path description, the file is searched for in VNCKs standard project folder VNC\_SIM.WPD. If the file shall be copied from a specific project folder, the complete path must be set.



### 3.3.8 Deleting a file in VNCK data management system

#### Asynchronous service

```
VNCDeleteProgram  
(  
    BSTR                sNcProgramPath,  
    long *              plActionId  
);
```

#### Event

```
SIMDeleteProgramResponse (  
    VNCResult_t*         ptResult,  
    long                 lActionId  
);
```

#### Parameters

ptResult	Resulting value
sNcProgramPath	Program path to delete
[p]lActionId	Identifier to callback functions

#### Description

Deletes a program in a VNCK project. After checking the parameters and the actual status of VNCK the response event will either report success or failure of a closed transfer.

### 3.3.9 Obtaining information about multi-file transfer

#### Event

```
SIMTransferFileInfo  
(  
  VNCRResult_t*          ptResult,  
  BSTR                   sPathSource,  
  BSTR                   sPathDest,  
  long *                  lActionId  
);
```

#### Parameters

ptResult	Resulting value
sPathSource	Next source file to be transferred
sPathDest	Next destination for file

#### Description

Information about the next file to be transferred when a command was started transferring a project or directory.

### 3.3.10 Obtaining information about transfer status

#### Event

```
SIMTransferFileStatus  
(  
  VNCRResult_t*          ptResult,  
  long                    lTransferRate,  
  long                    lActionId,  
);
```

#### Parameters

ptResult	Resulting value
lActionId	Identifier to callback functions
lTransferRate	percentual value of transfer rate
lActionId	Identifier to callback functions

#### Description

Information about the actual value of the data transfer from or to the VNCK. If the transfer is finished correctly there is a value of 100 percent for the transfer rate. If there are any failures the result parameter will provide information on the error.

---

#### Note

COM methods *SIMTransferFileInfo* and *SIMTransferFileStatus* are not implemented due to different behaviour of HMIAdvanced and SINUMERIK Operate. Both functions only give additional information and are not critical for customers.

---

## 3.4 Variable Services

### 3.4.1 General Information

The SINUMERIK 840D sl system handles several types of variables:

- System variables typically starting with '\$', e.g. \$P\_TOOL
- OEM and user defined variables

The access to SINUMERIK variables via VNCK COM interface is granted by three different access functions:

- Based on the original variable name shown e.g. in an NC part program
- Based on BTSS naming conventions
- Based on an extended BTSS interface using SINUMERIK internal identifier

All three access types are bound to certain restrictions, which are as follows.

#### **Using the NC part program representation of a variable/system variable**

This is the most common way to access variables. It works for system variables as well as for OEM or user defined variables.

The request is handled via private communication between VNCK server and the NCK system (compile cycle interface). There are no restrictions concerning the state of the NCK system, which means it works in freeze mode as well as during a VNCKRun action.

The VNCK COM functions VNCGetVariable, VNCSetVariable, VNCWatchVariable and VNCUnWatchVariable are used for this type of access.

#### **Using the standard BTSS naming convention**

This will only work for variables which are defined in the HMI system.

Please refer to the HMI documentation for further explanations, of how to address BTSS-Variables from HMI.

The usage of this type of access is only possible when the NCK system is not in freeze mode, since it is based on the standard communication between NCK and HMI. Between SIMFreeze and VNCKRun the NCK system is in a frozen state and will not be able to communicate with HMI.

The VNCK COM functions VNCGetVariable, VNCSetVariable, VNCWatchVariable and VNCUnWatchVariable are used for this type of access.

#### **Using the extended SINUMERIK internal BTSS naming convention**

This type of access is only for expert users, who are aware of the internal naming convention. It is based on the private communication between VNCK server and NCK, thus can be used also during freeze mode of NCK.

Variables of complex data types like FRAME and AXIS can only be accessed via this type of communication.

The VNCK COM function VNCReadBtssVariable is used for this type of access.

### 3.4.2 Reading Variables

#### Asynchronous service

```
VNCGetVariable  
(  
    long                IChannel,  
    BSTR                sVarName,  
    long *               plActionId  
);
```

#### Event

```
SIMGetVariableResponse  
(  
    VNCResult_t*        ptResult,  
    double               dVirtTime,  
    VARIANT *            pvValue,  
    long                 lActionId  
);
```

#### Parameters

ptResult	Resulting value
dVirtTime	Virtual time stamp
IChannel	Channel number
sVarName	Name of the variable to read
pvValue	Pointer to variable value
[p]lActionId	Identifier to callback functions

#### Description

This function enables the simulation to read the value of a SINUMERIK 840D sl defined variable in the VNCK system. Since the data type varies a variant data type is used for the resulting value parameter. If an array is described by the variable name pvValue points to a SAFEARRAY data type.

### 3.4.3 Writing Variables

#### Asynchronous service

VNCSetVariable	
(	
long	IChannel,
BSTR	sVarName,
VARIANT *	pvValue,
long *	plActionId
);	

#### Event

SIMSetVariableResponse	
(	
VNCResult_t*	ptResult,
double	dVirtTime,
long	lActionId
);	

#### Parameters

ptResult	Resulting value
dVirtTime	Virtual time stamp
IChannel	Channel number
sVarName	Name of the variable to write
pvValue	Pointer to variable value
[p]lActionId	Identifier to callback functions

#### Description

This function enables the simulation to write the value of a SINUMERIK 840D sl defined variable in the VNCK system. Since the data type varies a variant data type is used for the value parameter to be set. If an array is described by the variable name pvValue must point to a SAFEARRAY data type.

### 3.4.4 Watching Variables

#### Asynchronous service

```
VNCWatchVariable
(
    long                IChannel,
    BSTR                sVarName,
    long *              plActionId
);
```

```
VNCUnWatchVariable
(
    long *              plActionId
);
```

#### Event

```
SIMWatchVariableResponse
(
    VNCRResult_t*       ptResult,
    double              dVirtTime,
    long                lActionId
);
```

```
SIMUnWatchVariableResponse
(
    VNCRResult_t*       ptResult,
    double              dVirtTime,
    long                lActionId
);
```

```
SIMWatchVariableEvent
(
    double              dVirtTime,
    VARIANT *           pvValue,
    long                lVNCRunActionId,
    long                lActionId
);
```

#### Parameters

ptResult	Resulting value
dVirtTime	Virtual time stamp
IChannel	Channel number
sVarName	Name of the variable to watch
pvValue	Pointer to variable value
lVNCRunActionId	action ID of corresponding VNCRun call
[p]lActionId	Identifier to callback functions

## Description

This function enables the simulation to watch each change of the value of a SINUMERIK 840D sl defined variable in the VNCK system. Since the data type varies a variant data type is used for the value parameter. If an array is described by the variable name pvValue points to a SAFEARRAY data type.

---

### Note

After executing VNCKResetKernel watches to variables must be re-established.

---

### Note

In fact, if the function VNCUnWatchVariable() regards to a BTSS variable this function is implemented as a synchronous function. Therefore a warning value is returned to the request to advise to this behavior.

---

### Note

Behaviour when BTSS connection failes:

If the BTSS connction between HMI software and NCK is aborted for any season and afterwards re-established SIMWatchVariable events will be sent for all active watches. This behaviour is standard for real SINUMERIK controllers, but might be unwanted for simulation.

Adding the following lines to theVNC.ini will keep the BTSS connection alive also during freeze periods:

- [VNCK]
- activateBTSSwhileFreezing=1

This new setting is only available in VNCK4.5 or higher.

---

### Note

When watching BTSS variables the virtual time value is always 0.

If the exact virtual time information is needed, the according NCK system variable notation should be used (e.g. "\$P\_TOOL" instead of "/Channel/State/actTNumber").

### 3.4.5 Reading BTSS Variables

#### Asynchronous service

```
VNCReadBtssVariable
(
    long                IArea,
    long                IUnit,
    long                IInchMetric,
    long                IColIndex,
    long                IRowIndex,
    long                IModuleType,
    long                INumRows,
    long *              pIActionId
);
```

#### Event

```
SIMReadBtssVariableResponse
(
    VNCRResult_t *      ptResult,
    double              dVirtTime,
    long                INumber,
    unsigned char *      pcParam,
    long                IActionId
);
```

#### Parameters

ptResult	Resulting value
IArea	first BTSS variable parameter
IUnit	second BTSS variable parameter
IInchMetric	third BTSS variable parameter
IColIndex	forth BTSS variable parameter
IRowIndex	fifth BTSS variable parameter
IModuleType	sixth BTSS variable parameter
INumRows	seventh BTSS variable parameter
dVirtTime	Virtual time stamp
INumber	number of characters returned
pcParam	resulting character array
[p]IActionId	Identifier to callback functions

#### Description

This function enables the simulation in an expert mode to read the value of a SINUMERIK 840D sl defined variable in the VNCK system. In difference to VNCGet-Variable() this function is parameterized by the internal BTSS parameter setting. There is a SINUMERIK 840D sl reference book describing BTSS. The response event delivers a character array that must be interpreted at simulation side.

---

#### Note

There is another version of this event:  
SIMReadBtssVariableResponseSa ( ... );

---



## 3.5 VDI Services

### 3.5.1 General Information

These functions represent in a special view the absence of a real or virtual PLC. Thus it is possible to enable some behavior of the VNCK kernel in a way similar to PLC activity. Nevertheless there is no PLC logic running using this service. Neither a 'grundprogramm' or a 'anwenderprogramm' or emulations of these are running. Simulation is responsible itself for setting and clearing bits and bytes on the memory representing the VDI interface.

---

#### Note

If there is a VPLC process running the access to the VDI memory is restricted by the functions of IVNCServer.

---

### 3.5.2 Writing to the VDI Interface

#### Asynchronous service

```
VNCSetVDIVariable
(
    long                                IUnit,
    VDIVarType_t                       tVDIVarType,
    VDIVarValue_t                       tVDIVarValue,
    VARIANT *                           pvValue,
    long *                               plActionId
);
```

#### Event

```
SIMSetVDIVariableResponse
(
    VNCRResult_t*                       ptResult,
    long                                lActionId
);
```

#### Parameters

ptResult	Resulting value
IUnit	BAG, channel or axes number
tVDIVarType	Identifier of variable to set
tVDIVarValue	Defined enum value of variable to set
pvValue	Pointer to general value of variable to set
[p]lActionId	Identifier to callback functions

## Description

Using this function simulation is able to enable / disable different VNCK program processing modes that are determined by the value of the logical VDI interface. Since corresponding enum values are defined in `VDIVarValue_t` one of these values has to be used setting a variable identified by a value of `VDIVarType_t`.

---

### Note

In fact, this function is implemented as a synchronous function for most variable types. Therefore a warning value is returned to the request to advise to this behavior.

---

## 3.5.3 Reading from the VDI Interface

### Synchronous (partly asynchronous) service

```
VNCGetVDIVariable
(
    long                                IUnit,
    VDIVarType_t                       tVDIVarType,
    VARIANT *                           pvValue,
    long *                              plActionId
);
```

### Event

```
SIMGetVDIVariableResponse
(
    VNCResult_t*                       ptResult,
    VARIANT *                           pvValue,
    long                                lActionId
);
```

### Parameters

<code>ptResult</code>	Resulting value
<code>IUnit</code>	BAG, channel or axes number
<code>tVDIVarType</code>	Identifier of variable to read
<code>pvValue</code>	Pointer to general value of variable to be read
<code>[p]lActionId</code>	Identifier to callback functions

## Description

Using this function simulation is able to read values from the logical VDI interface. Since corresponding enum values are defined in `VDIVarValue_t` one of these values has to be used setting a variable identified by a value of `VDIVarType_t`.

---

**Note**

In fact, this function is implemented as a synchronous function for most variable types. Therefore a warning value is returned to the request to advise to this behavior.

If `VDIVarType_t VDI_VARTYPE_SINGLESTEP` is used, the function will be called asynchronously.

---

### 3.5.4 Handling FastIO via VDI Variables

There is a group of defines of the datatype `VDIVarType_t` that refers to all the bits and words of DB 10 that can be used to manage the digital and analogous input and output signals and values of a VNCK kernel from VDI side.

To simplify the identification of the enum values of the defines of `VDIVarType_t` they are based on the number of the first DBB to the according variable.

Please have a look to chapter '1.2 NCK I/O via PLC' of 'Function Manual Extended Functions 840S sl/828D'. Here the roles of all affected signals and values of the VDI interface are shown by diagrams. They demonstrate the ways between digital or analogous hardware input or output slots to the according system variables `$A_IN`, `$A_OUT`, `$A_INA` and `$A_OUTA`. When viewing analogous values please regard the influence of the slot specific weight factors.

---

**Note**

Writing to the digital or analogous output slots will not affect to the ncu since the ncu kernel overwrites these values in each IPO cycle.

---

### 3.5.5 Handling TSM mask in SINUMERIK Operate

SINUMERIK Operate offers the option to manually change tools, set spindle speeds and initiate specific M functions when the S840D controller is in JOG mode.

This option is implemented by predefined interactions between NC, PLC and HMI shown in the following figure.

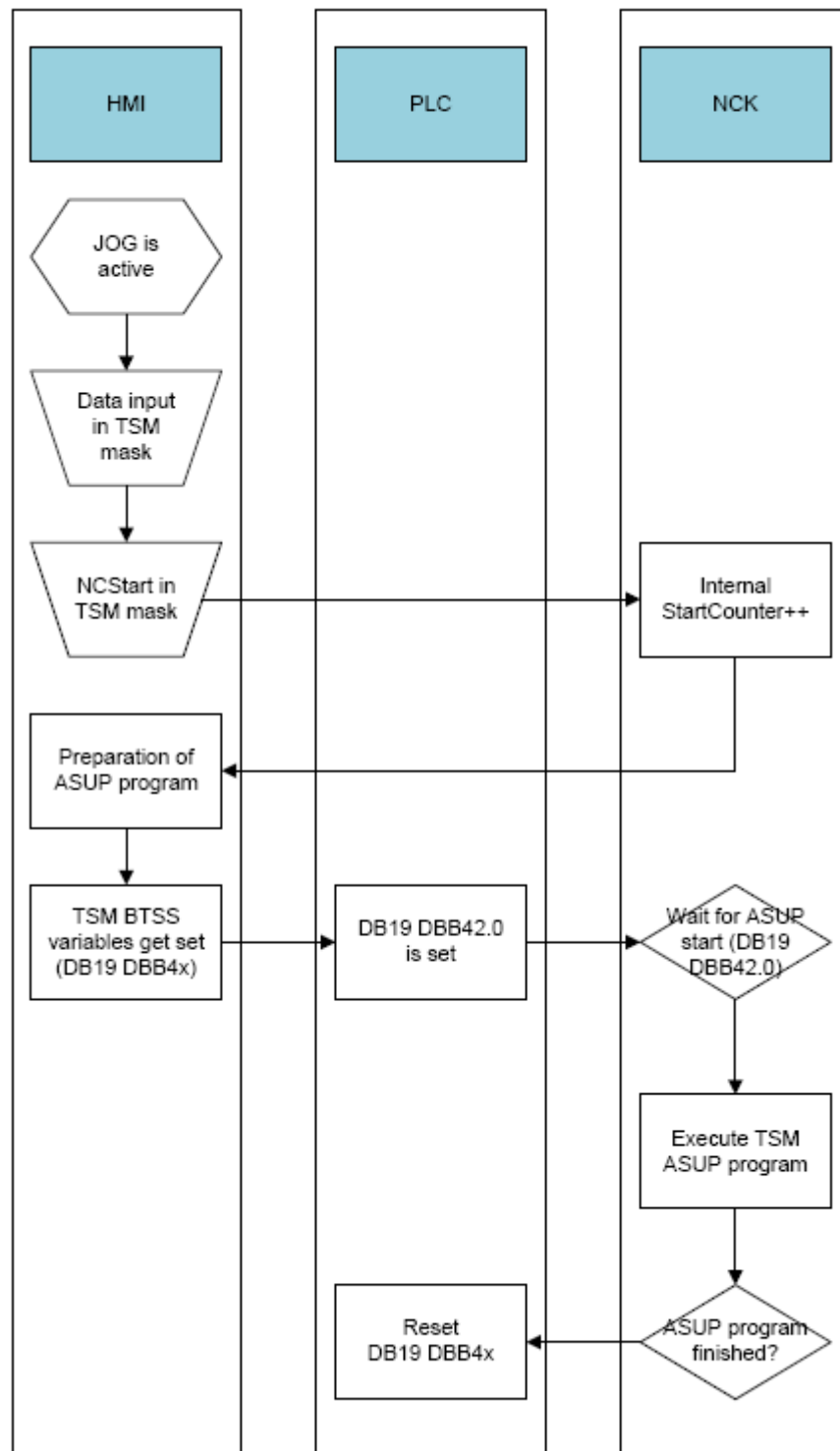


Figure 3-1: TSM interaction on real controller

The execution of an TSM program is initiated by HMI setting VDI variable DB19 DBB42.0 (and additional VDI variables) in the PLC via BTSS communication.

On a virtual S840D controller however the PLC controller part is not available, thus the initiation must be handled from the VNCK client by using the VNCK VDI interface as follows:

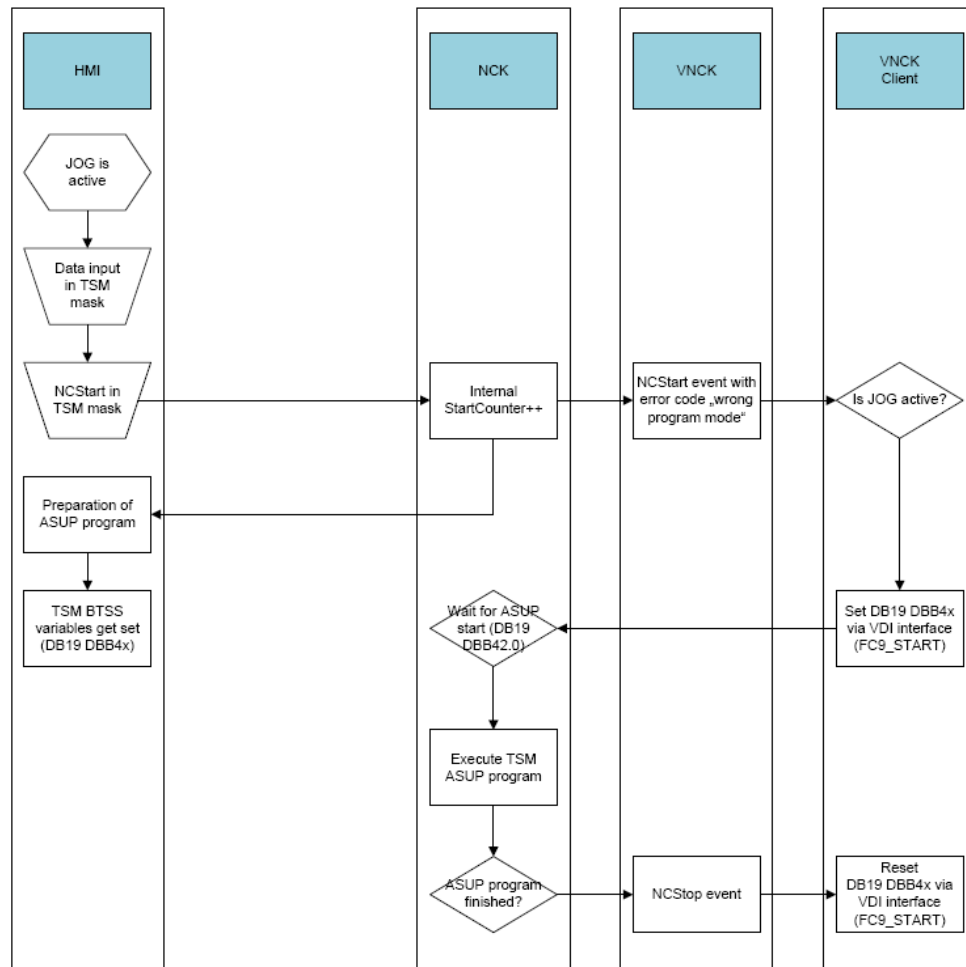


Figure 3-2: TSM interaction on virtual controller

With VNCK 4.5 SP1 an additional VDI variable type was introduced named VDI\_VARTYPE\_FC9\_START. With setting this variable via the VDI interface the needed DB19 DBB4x variables are set to initiate the execution of TSM programs.

#### Note

The existing function VNCProgStart was not modified, thus will send an error code VRV\_WRONG\_PROGRAM\_MODE when used in JOG mode.

## 3.6 VDI Fast Input and Output Services

### Synchronous services

```

VNCSetFastIOVariable
(
  VNCFastIOType_t          tIOType,
  long                     lIndex,
  VNCBooleanType_t        tDigitalValue,
  double                   dAnalogValue
);

VNCGetFastIOVariable
(
  VNCFastIOType_t          tIOType,
  long                     lIndex,
  VNCBooleanType_t *      ptDigitalValue,
  double *                 pdAnalogValue
);

VNCGetFastIONumberOfSlots
(
  VNCFastIOType_t          tIOType,
  long *                   plNumber
);

```

### Parameters

tIOType	Type describing the slot
lIndex	Number of selected slotType
[p]tDigitalValue	Boolean value of a digital slot
[p]dAnalogValue	Double value of an analogous slot
plNumber	Number of slots of the selected type

### Description

This functions VNCSetFastIOVariable (...) and VNCGetFastIOVariable (...) allows simulation to get access to an emulation of the slots of the fast input and output interface of the VNCK kernel. Depending of the tIOType describing the digital or analogous input or output interface either the paramter [p]tDigitalValue or [p]dAnalogValue is used to transfer the according slot value.

If the analogous slots are applied the values handled by the functions are computed with the according weight factors to each slot. That means simulation will read and can write just those analogous values that are visible with the according system variables \$A\_INA[slot] and \$A\_OUTA[slot]. Therefore the range of values is restricted to the negative and positive value to each slot weight. To avoid this automatically handling of the weight factors there is an entry in the file theVNC.ini:

```

[VNCK]
FastIOIgnoreWeightFactors=1

```

By default this line is not included to the file and there is a default value 0 ( zero ) to this entry.

The function VNCGetFastIONumberOfSlots (...) delivers the actual number of slots to the referred slot type. Of course this number is used as the border to the parameter lIndex of the functions VNCSetFastIOVariable (...) and VNCGetFastIOVariable (...).

---

**Note**

Since the access to the memory of emulation of the FastIO interface can be done directly when the interface function of the VNCK system is called the functions described in this chapter are implemented as synchronous functions. There are no according functions defined in any callback interface of the VNCK system. Therefore a warning value is returned to the request to advise to this behaviour.

---

## 3.7 VNCK Program Control by Slices

### 3.7.1 General Information

Program progress in VNCK is determined by a mode we call 'slicing'. This slice management is done to reduce the amount of data that is sent from VNCK while running a program. Otherwise the simulator can determine concrete times when VNCK hands over the control of program progress to the simulator. Before sending a 'SIMFreeze' event to do just this all path data set required setting the 'pathOutput options' are transferred.

The following slice modes can be activated:

**VNC\_SLICEMODE\_IPO\_TIME:**

A 'SIMFreeze' event is sent whenever the virtual time of interpolation has passed a given interval. The given time interval divided by the IPO cycle time determines the number of IPO cycles between the outputs of path data.

**VNC\_SLICEMODE\_LENGTH:**

A 'SIMFreeze' event is sent whenever the length of the path has reached a given distance to the last point of interruption. Thus the path can be scanned by checking it in geometric increments.

**VNC\_SLICEMODE\_ANGLE:**

A 'SIMFreeze' event is sent whenever the interpolation of a rotary axis has passed a given interval of degrees. Thus the processing of rotary axes can be scanned by checking it in geometric angle increments.

**VNC\_SLICEMODE\_ANGLE\_SPEED:**

A 'SIMFreeze' event is sent whenever the actual value of a rotary axis has passed a given interval of degrees. This mode addresses rotary axes that are not interpolated and running in speed mode. Thus the processing of rotary axes can be scanned by checking it in geometric angle increments.

**VNC\_SLICEMODE\_BLOCK\_CHANGED:**

A 'SIMFreeze' event is sent whenever a new block is changed into the internal task of interpolation. This mode allows the simulator to run a part program on a block-wise basis.

**VNC\_SLICEMODE\_SINGLE\_AXIS:**

A 'SIMFreeze' event is sent whenever a new block moving a single axis motion is changed into the internal task of interpolation. Thus commands as POS, POSA, SPOS, SPOSA etc. can be watched.

**VNC\_SLICEMODE\_SPINDLE\_SPEED:**

A 'SIMFreeze' event is sent whenever a new block changing the mode of a spindle under speed control is changed into the internal task of interpolation. Thus commands as M3, M4, M5 etc. can be watched.

**VNC\_SLICEMODE\_NOT\_INCYCLE:**

Activating this mode allows the VNCK kernel to run under slice controlling even if no part program has been started. This may be helpful e.g. if the operating mode JOG has been set and the change of axes values must be recognized.



All the criteria of 'slicing' can be added. For example you can order the block-Changed mode to get program line progress together with the length mode to keep contact to the path.

**VNC\_SLICEMODE\_KEEP\_CHECK\_VALUES:**

By default the actual values for time, length and angle slice are started with each SIMFreeze no matter what caused the SIMFreeze event. E.g. with time slice 100ms set and a SIMFreeze BlockChanged occurring after 50ms, the time slice calculation will start at value 0 with the next VNCRun. Setting this slice mode keeps the slice value counting to the original step size.

**VNC\_SLICEMODE\_FIPO\_CYCLE:**

A 'SIMFreeze' event is sent after each fine IPO cycle. Thus information on axes will be given in the most precise way that a SINUMERIK 840D sl can manage. This slice mode is for internal use only and requires a specific module license.

Furthermore, the simulator can control program progress by setting a 'freeze' mode. So, in addition to the slice mode, the VNCK will stop the current program at the end of the IPO cycle when a freeze condition is activated. In contrast to the slice mode the meeting of freeze mode criteria doesn't issue path output events.

The following freeze modes can be activated:

**VNC\_FREEZEMODE\_PROGRAM\_START:**

Whenever one channel starts program execution a 'SIMFreeze' event is sent at the end of the IPO cycle.

**VNC\_FREEZEMODE\_PROGRAM\_STOP:**

Whenever one channel stops program execution a 'SIMFreeze' event is sent at the end of the IPO cycle.

**VNC\_FREEZEMODE\_BLOCK\_CHANGED:**

Whenever a new block is changed into the internal task of interpolation a 'SIM-Freeze' event is sent at the end of the IPO cycle.

**VNC\_FREEZEMODE\_LEAVE\_SUBROUTINE:**

When a subroutine returns to the caller part program a 'SIMFreeze' event is sent at the end of the IPO cycle.

**VNC\_FREEZEMODE\_TOOL\_ACTION:**

Whenever a tool select or change or a tool carrier activation takes place, or when the tool correction changes mode or values a 'SIMFreeze' event is sent at the end of the IPO cycle.

**VNC\_FREEZEMODE\_REGISTERED\_COMMAND:**

When a registered command parameterized with freeze mode reaches the IPO task a 'SIMFreeze' event is sent at the end of the IPO cycle.

**VNC\_FREEZEMODE\_SINGLE\_AXIS:**

Whenever a new block moving a single axis motion is changed into the internal task of interpolation a 'SIMFreeze' event is sent at the end of the IPO cycle.

**VNC\_FREEZEMODE\_SPINDLE\_SPEED:**

Whenever a new block changing the mode of a spindle under speed control is changed into the internal task of interpolation a 'SIMFreeze' event is sent at the end of the IPO cycle.

**VNC\_FREEZEMODE\_MOTION\_CONTACT:**

Whenever a new block initiating a measuring movement or a drive to fixed stop is changed into the internal task of interpolation a 'SIMFreeze' event is sent at the end of the IPO cycle.

**VNC\_FREEZEMODE\_COLLISION\_GEO:**

Whenever a command WORKPIECE or FIXTURE is recognized at IPO time a 'SIMFreeze' event is sent at the end of the IPO cycle.  
This freeze mode is for internal use only and requires a specific module license.

Either when the slice or the freeze mode criteria becomes active, the same 'SIMFreeze()' event is fired.

---

**Note**

Whenever a 'SIMFreeze()' event has been received the simulator has to send a 'VNCRun()' command to continue the VNCK internal program progress controlled by the slice and freeze management described above.

Inside the 'frozen NCU' state the VNCK kernel doesn't consume any CPU usage time.

Furthermore the virtual real time of the VNCK system is not increased. Thus it is possible to let run a virtual clock describing correctly the same time that a real machine will spend on working. The parameter dVirtTime of many of the callback events delivering information of part program processing notifies just this time stamp representing a real machine time consumption.

---

### 3.7.2 Setting slice mode

#### Asynchronous service

```
VNCSetSliceMode
(
  VNCSliceType_t          tSliceMode,
  SliceValues_t*          ptSliceValues,
  long *                  plActionId
);
```

#### Event

```
SIMSetSliceModeResponse
(
  VNCResult_t*            ptResult,
  long                    lActionId
);
```

#### Parameter

ptResult	Resulting value
tSliceMode	Pattern describing the active slice modes
ptSliceValues	Array for options dependent on values
[p]lActionId	Identifier to callback functions

#### Description

Depending on the set pattern for the different slice modes these modes for event behavior are activated. For those modes depending on values to watch these values are initially set by the SliceValues\_t parameter. When the VNCK detects that at least one of the conditions is fulfilled, the ordered path output will take place. A time slice value is given in milliseconds. A length slice value is given in millimeters. Angle values are given in degrees.

### 3.7.3 Setting freeze mode

#### Asynchronous service

```
VNCSetFreezeMode
(
  VNCFreezeMode_t          tFreezeMode,
  long *                    plActionId
);
```

#### Event

```
SIMSetFreezeModeResponse
(
  VNCResult_t*              ptResult,
  long                      lActionId
);
```

#### Parameter

ptResult	Resulting value
tFreezeMode	Pattern describing the active freeze modes
[p]lActionId	Identifier to callback functions

#### Description

Depending on the set pattern these freeze modes for event behavior are activated. If VNCK found that one of the conditions is met at the end of an IPO cycle, this causes a SIMFreeze event additionally to the event caused by the freeze criterion.

### 3.7.4 Processing the next slice

#### Synchronous service

```
VNCRun
(
  long *                    plActionId
);
```

#### Parameters

plActionId	Identifier to callback functions
------------	----------------------------------

#### Description

After a preceding SIMFreeze() event this function resumes the program execution in the VNCK up to the next SIMFreeze() event.

---

#### Note

In fact, this function is implemented as a synchronous function. Therefore a warning value is returned to the request to advise to this behavior.  
The ActionID given as return value is used to refer to any callback functions sent between VNCRun and SIMFreeze.

---

### 3.7.5 Controller freeze

#### Event

```

SIMFreeze
(
  double                dVirtTime,
  VNCFreezeReason_t    tFreezeReason,
  long                 lActionId
);

```

#### Parameters

dVirtTime	Virtual time stamp
tFreezeReason	Reasons for Freeze
lActionId	lActionId of the preceding VNCRun()

#### Description

The VNCK informs the simulator of the transition to the frozen state by calling SIMFreeze. In this state, the VNCK no longer consumes virtual time. The VNCK usually changes to this state if any of the set slice conditions are met. Before SIMFreeze is called, the VNCK must transfer the current path data for all relevant channel or machine axes according to the ordered path output option. Only when the path data as well as further program information (i.e. the description of the notified help function causing the actual freeze event) have been completely transferred, the controller will signal the simulator by calling SIMFreeze that no additional process data for the previous time increment will be transmitted.

### 3.7.6 Timer Functions

#### Asynchronous service

```

VNCSetTimer
(
  double                dAlarmTime,
  VNCTimerMode_t       tTimerMode,
  long *                plActionId
);

```

#### Events

```

SIMSetTimerResponse
(
  VNCRResult_t *        ptResult,
  long                 lActionId
);

SIMTimerNotify
(
  double                dVirtTime,
  VNCTimerMode_t       tTimerMode,
  long                 lActionId
);

```

## Parameters

dAlarmTime	Time to be set the alarm
tTimerMode	Mode of timer handling
ptResult	Resulting value
dVirtTime	Virtual time stamp
[p]lActionId	Identifier to service or callback functions

## Description

The VNCSetTimer() function is used to activate an alarm clock. The callback function events are sent to response the order and to inform simulation once or several times whenever the timer conditions became true. The tTimerMode allows furthermore to activate an according SIMFreeze () event. Ordering a dAlarmTime with a value of zero stops the timer.

---

### Note

It is possible to manage several timers. Therefore the value of plActionId must be set to address an already installed timer. Not setting the value of plActionId creates a new timer parallel to eventually already running timers.

---

### Note

If dAlarmTime is set to values bigger than 80s the SIMTimerNotify event might be sent one IPO too late.

---

### Note

SIMFreeze events issued by VNCSetTimer may contain also additional freeze reasons like ProgramStop events.

---

## 3.8 NC Program Control Services

### 3.8.1 Program selection

#### Asynchronous service

```
VNCProgSelect
(
  long                                IChannel,
  VNCFileDescriptionMode_t           tFileDescType,      (= VNC_FILEDESC_PC)
  BSTR                               sFilePath,
  long *                             plActionId
);
```

#### Event

```
SIMProgSelectResponse
(
  VNCResult_t*                       ptResult,
  double                             dVirtTime,
  long                               IChannel,
  long                               lActionId
);
```

#### Parameters

ptResult	Resulting value
IChannel	Channel number
tFileDescType	Type of the path description
sFilePath	Path of NC program file
dVirtTime	Virtual time stamp
[p]lActionId	Identifier to callback functions

#### Description

Selects an NC program for processing in the VNCK. If no work piece directory is given in the program name, the default work piece is assumed to be the storage place for the program to be selected.

---

#### Note

The parameter tFileDescType must be set to 'VNC\_FILEDESC\_PC'.

---

### 3.8.2 Program selection for external processing

#### Asynchronous service

```
VNCProgSelectExtern  
(  
    long  
    VNCFileDescriptionMode_t  
    BSTR  
    long *  
);  
IChannel,  
tFileDescType,      (= VNC_FILEDESC_PC)  
sFilePath,  
pActionId
```

#### Event

```
SIMProgSelectExternResponse  
(  
    VNCResult_t*  
    double  
    long  
    long  
);  
ptResult,  
dVirtTime,  
IChannel,  
IActionId
```

#### Parameters

ptResult	Resulting value
IChannel	Channel number
tFileDescType	Type of the path description
sFilePath	Path of NC program file
dVirtTime	Virtual time stamp
[p]IActionId	Identifier to callback functions

#### Description

Using this function it is possible to process part programs that are not stored inside the VNCK kernel internal data management. Thus it is easy to process CAD programs or other programs that are too big to be downloaded to the size restricted store of the VNCK. Calling this function will make the VNCK server to register the external part program for processing, to select it as the actual active channel program and to download the program code in parts that are immediately processed.

---

#### Note

The parameter tFileDescType must be set to 'VNC\_FILEDESC\_PC'.

---



### 3.8.3 Enabling program execution

#### Asynchronous service

```
VNCProgStart
(
  long                                IChannel,
  long *                             plActionId
);
```

#### Event

```
SIMProgStartResponse
(
  VNCResult_t*                       ptResult,
  double                             dVirtTime,
  long                                IChannel,
  long                                lActionId
);

SIMProgStartEvent
(
  VNCResult_t *                       ptResult,
  double                             dVirtTime,
  long                                IChannel
);
```

#### Parameter

ptResult	Resulting value
dVirtTime	Virtual time stamp
IChannel	Channel number
[p]lActionId	Identifier to callback functions

#### Description

The service function allows the start of execution of the selected NC programs in the given channel. This command represents the pressing of the PLC Start button.

Whenever the VNCK detects that the channel is starting processing part programs it informs the simulator of the program start in the given channel. This must not be caused by simulations request. Also starting a part program via INIT command from another channel, the progEvent feature of the VNCK or ASUP facilities can cause program starts in the VNCK kernel.

---

#### Note

The callback event SIMProgStartEvent () is the explicit event that tells simulation about that the channel has reached a state 'Processing Part Program'.

---

---

#### Note

SIMProgStartResponse will send the warning value VRV\_CHANNEL\_COMMAND\_WARNING\_BY\_STOP\_CONDITION, if BTSS variable /channel/state/stopcond[u] does not have value 0. Values different from 0 signal that the channel rests in a specific wait condition. The wait conditions and their meaning is documented in DocOnWeb. An excerpt is shipped as file channelStateStopcond.txt in VNCK's doc folder

---

### 3.8.4 Stopping program execution

#### Asynchronous service

```
VNCProgStop
(
    long                IChannel,
    long *              plActionId
);
```

#### Events

```
SIMProgStopResponse
(
    VNCResult_t*        ptResult,
    double               dVirtTime,
    long                IChannel,
    long                IActionId
);

SIMProgStopEvent
(
    double               dVirtTime,
    long                IChannel,
    VNCProgStopType_t   tProgStopType
    long                IVNCRunActionId,
);
```

#### Parameter

ptResult	Resulting value
dVirtTime	Virtual time stamp
IChannel	Channel number
tProgStopType	Reason for Stopping
IVNCRunActionId	IActionId of the preceding VNCRun()
[p]IActionId	Identifier to callback functions

## Description

The service function interrupts program execution in the given channel. Program execution can be resumed from this point with VNCProgStart. The conditions of the slice control are reset.

Whenever the VNCK detects that the channel is stopping processing part programs it informs the simulator of the program stop in the given channel. As well the dVirtTime and the tProgStopType parameter are describing the time and the reason for this change of channel state.

---

### Note

The callback event SIMProgStopEvent() is the explicit event that tells simulation about that the channel has reached a state 'NOT Processing Part Program'.

---

### Note

SIMProgStopResponse will send a warning value VRV\_CHANNEL\_COMMAND\_WARNING\_BY\_STOP\_CONDITION, if BTSS variable /channel/state/stopcond[u] does not have value 0. Values different from 0 signal that the channel rests in a specific wait condition. The wait conditions and their meaning is documented in DocOnWeb. An excerpt is shipped as file channelStateStopcond.txt in VNCK's doc folder.

---

## 3.8.5 Resetting program execution

### Asynchronous service

```
VNCProgReset
(
    long                IChannel,
    long *              plActionId
);
```

### Event

```
SIMProgResetResponse
(
    VNCResult_t*        ptResult,
    double              dVirtTime,
    long                IChannel,
    long                lActionId
);
```

```
SIMProgResetEvent
(
    VNCResult_t *       ptResult,
    double              dVirtTime,
    long                IChannel
);
```

## Parameter

ptResult	Resulting value
dVirtTime	Virtual time stamp
IChannel	Channel number
[p]IActionId	Identifier to callback functions

## Description

The service function aborts program execution in the given channel. Program execution cannot be resumed from this point of the part program. The conditions of the slice control are reset.

Whenever the VNCK detects that the channel is resetting processing part programs it informs the simulator of the program stop in the given channel.

---

### Note

The callback event SIMProgStopEvent() is the explicit event that tells simulation about that the channel has reached a state 'NOT Processing Part Program'.

---

### Note

SIMProgResetResponse will send a warning value VRV\_CHANNEL\_COMMAND\_WARNING\_BY\_STOP\_CONDITION, if BTSS variable /channel/state/stopcond[u] does not have value 0. Values different from 0 signal that the channel rests in a specific wait condition. The wait conditions and their meaning is documented in DocOnWeb. An excerpt is shipped as file channelStateStopcond.txt in VNCK's doc folder.

---

## 3.9 Extended Program Control Services

### 3.9.1 General Info

There are several services VNCK system offers to simulation to take influence in processing part programs. As well at interpretation time lines of part program code can be changed as at interpolation time events are sent reporting the processing of registered commands. Thus this function provides an opportunity for the simulator to carry out its own actions at interpolation time. Furthermore functions are implemented to modify channel interpolation activity. Thus eventually waiting for PLC confirmations can be emulated within the VNCK system.

### 3.9.2 Registering Patterns for Interpretation

#### Asynchronous services

```
VNCRegisterPattern
(
    long                                IChannel,
    BSTR                                sPattern,
    long *                              plActionId
);

VNCUnRegisterPattern
(
    long                                IChannel,
    long                                lActionId
);

VNCTranslationContinue
(
    long                                lActionId
);

VNCReplaceLine
(
    BSTR                                sNewProgramLine,
    long                                lActionId
);
```

## Event

```

SIMRegisterPatternResponse
(
  VNCRResult_t *          ptResult,
  long                    lActionId
);

SIMUnRegisterPatternResponse
(
  VNCRResult_t *          ptResult,
  long                    lActionId
);

SIMPatternNotify
(
  VNCRResult_t *          ptResult,
  long                    lChannel,
  long                    lLineNumber,
  BSTR                    sLine,
  BSTR                    sProgramName,
  long                    lActionId
);

```

## Parameters

ptResult	Resulting value
lChannel	Channel number
sPattern	Pattern to find and check
sNewProgramLine	New part program line
lLineNumber	Number of line containing found pattern
sLine	Line containing found pattern
sProgramName	Name of program containing found pattern
lVNCRRunActionId	lActionId of the preceding VNCRRun()
[p]lActionId	identifier of registered command

## Description

Using VNCRRegisterPattern the simulator can request notifications of part program lines before these lines are translated and interpreted in the VNCK internal preparation task. In response to the notification the simulation must either send a changed new program line to be interpreted or simulation must order to use the original line. Using this function simulation can watch processing part program in aspect to patterns that may be changed for special simulation requests. This mechanism enables the simulator perhaps to correct programs that are based on settings that doesn't work since no PLC functionality or other external periphery is implemented in the VNCK.

---

### Note

Be very careful using this function. VNCK system can't take any responsibility to changed part processing based on simulation manipulated program code.

---

### 3.9.3 Executing registered NC commands

#### Asynchronous service

```
VNCRegisterCommand
(
    long                                IChannel,
    VNCRegCmdType_t                    tRegCmdType,
    VNCRegCmdDescType_t *              ptRegCmdDesc,
    long *                              plActionId
);

VNCUnRegisterCommand
(
    long                                IChannel,
    long                                IActionId
);
```

#### Event

```
SIMRegisterCommandResponse
(
    VNCResult_t*                       ptResult,
    long                                IActionId
);

SIMUnRegisterCommandResponse
(
    VNCResult_t*                       ptResult,
    long                                IActionId
);

SIMCommandNotify
(
    double                             dVirtTime,
    long                                IChannel,
    VNCRegCmdType_t                    tRegCmdType,
    VNCRegCmdDescription_t *           ptRegCmdDesc,
    long                                IVNCRunActionId,
    long                                IActionId,
);
```

#### Parameters

ptResult	Resulting value
IChannel	Channel number
tRegCmdType	Type of the command to register
ptRegCmdDesc	Parameters of the command to be registered or that was found in program progress
dVirtTime	Virtual time stamp
IVNCRunActionId	IActionId of the preceding VNCRun()
[p]IActionId	Identifier of registered command

## Description

Using VNCRegisterCommand, the simulator can request notifications for the execution of individual NC commands from interpolation in the VNCK. Setting the freezeMode parameter of the command description VNCK will interrupt execution of the actual slice. There are several types of commands to be registered:

### VNC\_REGISTER\_CMD\_FCT:

Auxiliary functions as M or H as well as further address commands such as S or F can be registered. The ignore Type parameter determines whether the extension and / or the value of the function must match the given integer or double values. Setting the ignore flags all functions independent from these values will be reported.

### VNC\_REGISTER\_CMD\_PATTERN:

The occurrence of a programmed pattern can be registered to force an event at ipo time when the related program line is executed. Pattern means any sequence of letters and digits written in part program code. The value parameters are not relevant.

### VNC\_REGISTER\_CMD\_LABEL:

The occurrence of a programmed label can be registered to force an event at ipo time when the related program line is executed. A label is sequence of letters and digits followed by a colon to identify a line of code in part program similar to a line number. The ignore Type and value parameters are not relevant.

The register request can be cancelled with VNCUnRegisterCommand and the matching ActionID received with VNCRegisterCommand.

When the VNCK starts executing a registered command, it will inform the simulator of this fact by calling the SIMCommandNotify function. This mechanism enables the simulator to simulate activities within the working space of the machine that do not result from motions of the actual NC axes (e.g. head change, pallet change). This is especially required in cases where no PLC functionality is implemented in the VNCK.

---

#### Note

No wildcards are allowed on using registering labels and patterns.

---

#### Note

Functions of type VNC\_REGISTER\_CMD\_FCT may be programmed as 'quick' functions to affect the handling of them inside the VNCK kernel. E.g. there is a program line 'N100 M=QU(4711)'. To inform simulation about this 'quick' type an additional bit VNC\_REGISTER\_CMD\_FCT\_QUICK is set to the parameter tRegCmdType of the event SIMCommandNotify(...).

In standard case (M=4711; no quick function) SIMCommandNotify is sent with value VNC\_REGISTER\_CMD\_NONE.

---

#### Note

If VNCRegisterCommand/VNCUnRegisterCommand is called with channel 0, the command description is (un-)registered with all existing channels.

---



### 3.9.4 Enabling Path Interpolation

#### Asynchronous services

```
VNCBreakChannel
(
  long                                IChannel,
  long *                             plActionId
);
```

```
VNCContinueChannel
(
  long                                IChannel,
  long                                lActionId
);
```

#### Event

```
SIMBreakChannelResponse
(
  VNCRResult_t*                      ptResult,
  long                                lActionId
);
```

```
SIMContinueChannelResponse
(
  VNCRResult_t*                      ptResult,
  long                                lActionId
);
```

#### Parameter

PtResult	Resulting value
IChannel	Channel number
[p]lActionId	Identifier to callback functions

#### Description

Using these functions simulation is able to break and continue IPO processing of one channel just like a confirmation of a not exiting PLC function acknowledgement is missing. The break of interpolation is based on setting VNCK kernel internal flags controlling curve interpolation, positioning activity, spindle status and ipo block changes. Using the break function will show how the VNCK kernel will break and slow down all active channel axes to zero movement just as if a PLC confirmation is missing.

## 3.10 Path Data Services

### 3.10.1 Setting path data output option

#### Synchronous service

```
VNCSetPathOutputOption
(
    long                                IChannel,
    VNCPathOutputOption_t              tPathOutputOption,
    long *                              plActionId
);
```

#### Event

```
SIMSetPathOutputOptionResponse
(
    VNCRResult_t*                      ptResult,
    long                               ActionId
);
```

#### Parameter

ptResult	Resulting value
IChannel	Channel number
tPathOutputOption	Pattern describing the data records to send to simulation
[p]lActionId	Identifier to callback functions

#### Description

All the options will activate separate events of information transfer from VNCK to the simulator at interpolation time. These events will be sent in order just before a SIMFreeze() event is sent either because a slice is closed or because an other reason breaks VNCK part program processing.

To understand the parameters of VNCPathOutputOption\_t see the following figure:

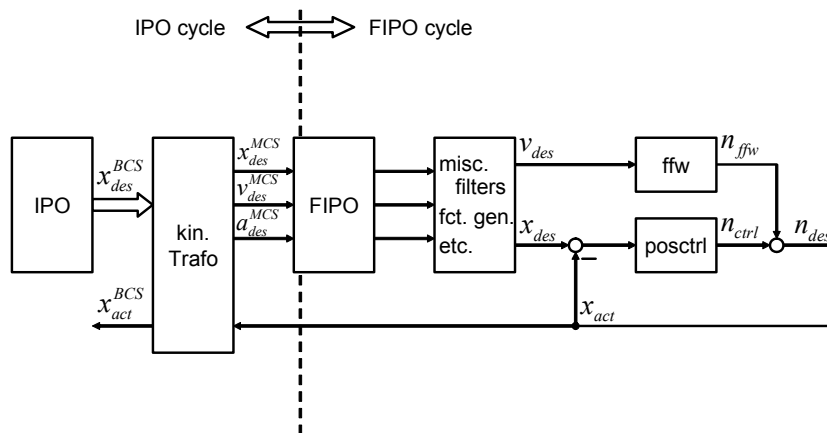


Figure 3-3: Parameters of VNCPathOutputOption\_t

VNCPathOutputOption_t	symbol
VNC_PATH_OUTPUT_BCS	$x_{des}^{BCS}$
VNC_PATH_OUTPUT_FEED	$v_{des}^{MCS}$
VNC_PATH_OUTPUT_ACC	$a_{des}^{MCS}$
VNC_PATH_OUTPUT_MCS	$x_{des}^{MCS}$
VNC_PATH_OUTPUT_BCS_ACTUAL_POS	$x_{act}^{BCS}$
VNC_PATH_OUTPUT_MCS_ACTUAL_POS	$x_{act}$
VNC_PATH_OUTPUT_MCS_CMD_POS	$x_{des}$
VNC_PATH_OUTPUT_MCS_CMD_VEL	$v_{des}$
VNC_PATH_OUTPUT_MCS_CMD_VEL_TO_DRIVE	$n_{des}$
VNC_PATH_OUTPUT_TOOL_CP	not shown
VNC_PATH_OUTPUT_TOOL_ORI	not shown
VNC_PATH_OUTPUT_TOOL_VEL	not shown
VNC_PATH_OUTPUT_BCS2	not shown
VNC_PATH_OUTPUT_WCS	not shown
VNC_PATH_OUTPUT_SZS	not shown
VNC_PATH_OUTPUT_BZS	not shown

Setting the path output option VNC\_PATH\_OUTPUT\_LAST\_VALUES\_ON\_MC in combination with the listed options above forces the VNCK system to issue additional path output callbacks each time the executed block changes. The callback sends information on the last path position before the actual block change takes place. Thus simulation is enabled to identify the exact position before and after the block change.

#### Note

If VNCSetPathOutputOption is called with channel 0, the path output is registered with all existing channels.

#### Note

The following path output options are not available with the standard VNCK license. These options are restricted to Siemens internal usecases:

VNC\_PATH\_OUTPUT\_BCS\_ACTUAL\_POS  
VNC\_PATH\_OUTPUT\_MCS\_ACTUAL\_POS  
VNC\_PATH\_OUTPUT\_MCS\_CMD\_POS  
VNC\_PATH\_OUTPUT\_MCS\_CMD\_VEL  
VNC\_PATH\_OUTPUT\_MCS\_CMD\_VEL\_TO\_DRIVE

### 3.10.2 Getting path data output events

#### Asynchronous service

```
VNCGetPathOutput
(
    long                                IChannel,
    long *                             plActionId
);
```

#### Event

```
SIMGetPathOutputResponse
(
    VNCResult_t*                       ptResult,
    long                               ActionId
);
```

#### Parameter

ptResult	Resulting value
IChannel	Channel number
[p]lActionId	Identifier to callback functions

#### Description

Calling VNCGetPathOutput () will issue all described path output events after the next ipo cycle has been passed inside the VNCK kernel.

### 3.10.3 Sending path output data

There are several different event functions to transfer the requested path output values to the simulator.

#### Events for path output information

```
SIMPathOutput
(
    Double                             dVirtTime,
    long                               IChannel,
    VNCPathOutputOption_t             tPathOutputOption,
    long                               INumber,
    VNCAxesMode_t *                   ptAxesMode,
    double *                           pdValue,
    long                               IVNCRunActionId
);
```

## Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
tPathOutputOption	Identifies the type of path output
INumber	Number of data sets in array
ptAxesMode	Array of axes modes
pdValue	Array of axes position values
IVNCRunActionId	Identifier from VNCRun()

## Description

Using these events the VNCK tells the simulator the actual values of ordered path data records describing the actual state of VNCK program progress.

---

### Note

There are other versions of these events:  
SIMPathOutputSa ( ... );

---

## 3.10.4 Handling Collision Limits

### Asynchronous service

```
VNCSetCollisionLimit
(
  VNCCheckCollisionLimitMode_t  tCCLMode,
  long                           IMachineAxisIndex,
  double                         dLimitMinus,
  double                         dLimitPlus,
  long *                         plActionId
);
```

### Event

```
SIMSetCollisionLimitResponse
(
  VNCKResult_t *                ptResult,
  long                          lActionId
);
```

### Parameter

ptResult	Resulting value
tCCLMode	Definition of the limit type
IMachineAxisIndex	Index of the applied machine axis
dLimitMinus	Negative axes value limit
dLimitPlus	Positive axes value limit
[p]lActionId	Identifier to callback functions

## Description

Using this function simulation can define axes position values where the VNCK kernel retains these values. This means the actual position of the axes doesn't change anymore. In this way VNCK system offers the facility to simulation to define the axes positions where a programmed FXS(...) command inside the NC part program reaches the fixed stop state.

---

### Note

Using the functionality is restricted by a Siemens internal license option. The function is obsolete since function VNCMotionContact is available.

---

## 3.10.5 Setting Actual MCS Axes Positions

### Asynchronous service

```
VNCSetMcsActPos  
(  
    long                INumber,  
    long *              plMachineAxisIndex,  
    double *            pdMCSAxValue,  
    long *              plActionId  
);
```

### Event

```
SIMSetMCSActPosResponse  
(  
    VNCResult_t *      ptResult,  
    long               lActionId  
);
```

### Parameter

ptResult	Resulting value
INumber	Number of axes to set
plMachineAxisIndex	Array of axis indices to be set
pdMCSAxValue	MCS axes values to be set
[p]lActionId	Identifier to callback functions

## Description

Using this function simulation can set actual MCS axes positions. INumber describes the number of the list of machine axes to be set.

---

### Note

Using the functionality is restricted by a Siemens internal license option.

---

---

**Note**

There is another version of this function:  
VNCSetMcsActPosSa ( ... );

---

## 3.11 Program Data Services

### 3.11.1 Setting program data output option

#### Synchronous service

```
VNCSetProgOutputOption  
(  
    long                                IChannel,  
    VNCProgOutputOption_t              tProgOutputOption,  
    long *                             plActionId  
);
```

#### Event

```
SIMSetProgOutputOptionResponse  
(  
    VNCResult_t*                       ptResult,  
    long                               ActionId  
);
```

#### Parameter

ptResult	Resulting value
IChannel	Channel number
tProgOutputOption	Pattern describing the data records to send to simulation
[p]IActionId	Identifier to callback functions

#### Description

All the options will activate separate events of information transfer from VNCK to the simulator at interpolation time. These events will be sent when a new block containing data of the ordered type is handled by the interpolation task for the first time. This means the time shows the first ipo cycle the block is in process of interpolation.

---

**Note**

If VNCSetProgOutputOption is called with channel 0, the prog output is registered with all existing channels.

---

### 3.11.2 Program display

#### Event

SIMCurrentProgramLine	
(	
double	dVirtTime,
long	IChannel,
long	ILineNumber,
BSTR	sLine,
BSTR	sProgramName,
long	IVNCRunActionId
);	

#### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
sLineNumber	Line offset in the ascii text file
sLine	Current line in part program
sProgramName	Current program name
IVNCRunActionId	IActionId of the preceding VNCRun()

#### Description

SIMCurrentProgramLine transmits the currently executed program line of the part program as well as its position in the part program.

### 3.11.3 User program message

#### Event

SIMCurrentMessage	
(	
double	dVirtTime,
long	IChannel,
BSTR	sMessage,
long	IVNCRunActionId
);	

#### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
sMessage	Message in part program
IVNCRunActionId	IActionId of the preceding VNCRun()

#### Description

SIMCurrentMessage transmits the value of the VNCK variable containing the last programmed user message. This programming is done by writing MSG("text") in the part program. The event is used in the same way to report the clearing of the message by MSG("").



### 3.11.4 Beginning of a new motion

#### Event

```

SIMNewMotionLin
(
  double
  long
  VNCMotionDataLin_t *
  long
);

SIMNewMotionCircle
(
  double
  long
  VNCMotionDataCircle_t *
  long
);

SIMNewMotionSpline
(
  double
  long
  VNCMotionDataSpline_t *
  long
);

SIMNewMotionLinAllAxes
(
  double
  long
  VNCProgOutputOption_t
  VNCMotionDataBasicInfo_t*
  long
  VNCAxesMode_t *
  double *
  double *
  long
);

SIMNewMotionCircleAllAxes
(
  double
  long
  VNCProgOutputOption_t
  VNCMotionDataBasicInfo_t*
  VNCMotionDataCircleInfo_t*
  long
  VNCAxesMode_t *
  double *
  double *
  long
);

```

dVirtTime,  
 IChannel,  
 ptMotionDataLin,  
 IVNCRunActionId  
  
 dVirtTime,  
 IChannel,  
 ptMotionDataCircle,  
 IVNCRunActionId  
  
 dVirtTime,  
 IChannel,  
 ptMotionDataSpline,  
 IVNCRunActionId  
  
 dVirtTime,  
 IChannel,  
 tProgOutputOption,  
 ptVNCMoDaBasicInfo,  
 INumber,  
 ptAxesMode,  
 pdAxValueStart,  
 pdAxValueTarget,  
 IVNCRunActionId  
  
 dVirtTime,  
 IChannel,  
 tProgOutputOption,  
 ptVNCMoDaBasicInfo,  
 ptVNCMoDaCircleInfo,  
 INumber,  
 ptAxesMode,  
 pdAxValueStart,  
 pdAxValueTarget,  
 IVNCRunActionId

```

SIMNewMotionSplineAllAxes
(
double                                dVirtTime,
long                                  IChannel,
VNCProgOutputOption_t                tProgOutputOption,
VNCMotionDataBasicInfo_t*            ptVNCMoDaBasicInfo,
long                                  INumber,
VNCAxesMode_t *                      ptAxesMode,
double *                             pdAxValueStart,
double *                             pdAxValueTarget,
long                                  IVNCRunActionId
);

```

### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
tProgOutputOption	Type of axes data
ptMoDaLin	Description of the new linear motion
ptMoDaCircle	Description of the new circular motion
ptMoDaSpline	Description of the new spline motion
ptVNCMoDaBasicInfo	Description of basic motion data
ptVNCMoDaCircleInfo	Description of circle motion data
INumber	Count of following axes values
pdAxValueStart	Array of axes values at block start
pdAxValueTarget	Array of axes values at block target
IVNCRunActionId	IActionId of the preceding VNCRun()

### Description

The VNCK sends these events if the interpolator has started a new motion toward a programmed position. Depending on the type of motion one of the events containing the appropriate data of the motion is sent. In this way the simulation is able to plan visualization of the motion and to recalculate the removal of material up to the end of the motion.

With the SIMNewMotionXxxAllAxes() events additionally the start and target axes values of all axes are sent including axes, which are not part of the programmed motion. The coordinate system, in which the target positions are expressed can be set via VNCSetProgramOutput() and the parameter:

VNC\_PROG\_OUTPUT\_NEW\_TARGET\_TCP:  
TCP positions of GEO axes, BCS positions of all other channel axes

VNC\_PROG\_OUTPUT\_NEW\_TARGET\_BCS:  
BCS positions of all channel axes

VNC\_PROG\_OUTPUT\_NEW\_TARGET\_MCS:  
MCS positions of all channel axes

---

### Note

There are other versions of these events:  
SIMNewMotionLinAllAxesSa ( ... );  
SIMNewMotionCircleAllAxesSa ( ... );  
SIMNewMotionSplineAllAxesSa ( ... );.

---

## 3.11.5 Single Axis Motion Management

### Events

```

SIMNewSingleAxisMotion
(
double                                dVirtTime,
long                                  IChannel,
long                                  IChanAxIndex,
VNCSingleAxisMotionType_t            tSAMType,
VNCSingleAxisMotionData_t *          ptSAMData,
long                                  ISAMId,
long                                  IVNCRunActionId
);

SIMEndSingleAxisMotion
(
double                                dVirtTime,
long                                  IChannel,
long                                  IChanAxIndex,
long                                  ISAMId,
long                                  IVNCRunActionId
);

```

### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
IChanAxIndex	Index of the channel axis
tSAMType	Type of starting motion
ptSAMData	Parameters describing the motion
ISAMId	Identifier to actual starting motion
IVNCRunActionId	IActionId of the preceding VNCRun()

### Description

These messages are reporting the start and end of a single axis motions.  
Single axis motions are:

- Spindle motions initiated by M commands like M3, M4, M5
- Motions of NC axes initiated by POS or POSA
- Motions of NC axes initiated by PLC functions (only if VPLC exists).

The parameter ISAMId can be used to assign the endOfMotion event to the startOfMotion event.

---

**Note**

If a single axis motion of a spindle is tracked and the spindle is stopped by command M5, the matching event is issued as soon as the spindle reached speed zero. In combination with program stop (M0) or program end (M30) the event might be issued after the program stop event. The event is not issued, if slice mode VNC\_SLICEMODE\_NOT\_INCYCLE is inactive.

---

### 3.11.6 IPO Block Change

#### Event

```
SIMBlockChanged  
(  
  double                dVirtTime,  
  long                  IChannel,  
  long                  IVNCRunActionId  
);
```

#### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
IVNCRunActionId	ActionId of the preceding VNCRun()

#### Description

This message is sent as soon as the interpolator changes the actually processed NC block.

### 3.11.7 Tool selection

#### Event

```
SIMToolSelect
(
  double          dVirtTime,
  long            IChannel,
  BSTR           sToolId,
  long            IToolHolder,
  double          dToolGeold,
  long            IVNCRunActionId
);
```

#### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
sToolId	Tool Identifier
IToolHolder	ID of affected toolHolder
dToolGeold	ID of geometric data of the tool
IVNCRunActionId	IActionId of the preceding VNCRun()

#### Description

Sent from VNCK at selection of the tool. The identifier describes either the tool number or the tool name depending on the mode of tool management in VNCK. If the identifier for the geometrical description exists it is sent as well. Depending on the mode of tool management the parameter IToolHolder either results from a programmed toolHolder or spindle or from the actual masterToolHolder or masterSpindle.

---

#### Note

In case a T command is programmed with extension IToolHolder reflects the programmed extension value. Otherwise the following rule is used to determine the active holder or spindle:

Regardless if tool management is active or not the system variable \$AC\_TC\_MTHNUM is used to determine the active holder or spindle.

If tool management is active and tool holders are activated then

\$AC\_TC\_MTHNUM reflects the last programmed SETMTH, otherwise it reflects the last programmed SETMS.

---

### 3.11.8 Tool change

#### Event

```
SIMToolChange  
(  
  double          dVirtTime,  
  long            IChannel,  
  BSTR            sToolId,  
  long            IToolHolder,  
  long            INumber,  
  VNCToolOffset_t* ptToolOffsets,  
  long            IVNCRunActionId  
);
```

#### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
sToolId	Tool ID
IToolHolder	ID of affected toolHolder
INumber	Number of tool offset data sets
ptToolOffsets	Tool offset records
IVNCRunActionId	IActionId of the preceding VNCRun()

#### Description

This call is sent by the VNCK at the beginning of the tool change action. It delivers the name of the new tool with a set of all its offsets.  
Depending on the mode of tool management the parameter IToolHolder either results from a programmed toolHolder or spindle or from the actual masterToolHolder or masterSpindle.

---

#### Note

In case a T command is programmed with extension IToolHolder reflects the programmed extension value. Otherwise the following rule is used to determine the active holder or spindle:  
Regardless if tool management is active or not the system variable \$AC\_TC\_MTHNUM is used to determine the active holder or spindle.  
If tool management is active and tool holders are activated then \$AC\_TC\_MTHNUM reflects the last programmed SETMTH, otherwise it reflects the last programmed SETMS.

---

#### Note

There is an other version of this event:  
SIMToolChangeSa ( ... );

---

### 3.11.9 Selecting a new tool offset

#### Event

```

SIMToolOffset
(
double
long
BSTR
long
VNCToolOffset_t*
VNCToolPlane_t
long
);
dVirtTime,
IChannel,
sToolId,
IToolHolder,
ptToolOffset,
tWorkingPlane,
IVNCRunActionId

```

#### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
sToolId	Tool ID
IToolHolder	ID of affected toolHolder
ptToolOffsets	Current tool offset record
tWorkingPlane	Current working Plane G17 / G18 / G19
IVNCRunActionId	IActionId of the preceding VNCRun()

#### Description

With this call the simulator receives information about the new active tool offset and the active tool plane.

Depending on the mode of tool management the parameter IToolHolder either results from a programmed toolHolder or spindle or from the actual masterToolHolder or masterSpindle.

---

#### Note

Regardless if tool management is active or not the system variable \$AC\_TC\_MTHNUM is used to determine the active holder or spindle. If tool management is active and tool holders are activated then \$AC\_TC\_MTHNUM reflects the last programmed SETMTH, otherwise it reflects the last programmed SETMS.

---

### 3.11.10 Tool Carrier Selection

#### Event

SIMToolCarrier	
(	
double	dVirtTime,
long	IChannel,
long	IToolCarrierNumber,
long	IVNCRunActionId
);	

#### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
IToolCarrierNumber	Number of the active tool carrier
IVNCRunActionId	IActionId of the preceding VNCRun()

#### Description

Sent from VNCK at selection / activation of a tool carrier.

### 3.11.11 Subroutine call

#### Event

SIMCallSubroutine	
(	
double	dVirtTime,
long	IChannel,
long	ICallStackIndex,
BSTR	sProgramName,
long	IVNCRunActionId
);	

#### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
ICallStackIndex	Index on call stack of called subroutine
sProgramName	Name of called subroutine
IVNCRunActionId	IActionId of the preceding VNCRun()

#### Description

The VNCK always sends this event when the program execution enters a subroutine. The call stack index describes the actual subprogram level entered with this subprogram. SIMCallSubroutine will be sent for all subroutine calls even if several levels of subprograms do not contain any interpolation block information. Thus several SIMCallSubroutine calls can be sent successively.



### 3.11.12 Return from subroutine

#### Event

SIMLeaveSubroutine	
(	
double	dVirtTime,
long	IChannel,
long	ICallStackIndex,
BSTR	sProgramName,
long	IVNCRunActionId
);	

#### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
ICallStackIndex	Index on call stack of closed subroutine
sProgramName	Name of closed subroutine
IVNCRunActionId	IActionId of the preceding VNCRun()

#### Description

The VNCK always sends this event when the program execution leaves a subprogram. SIMLeaveSubroutine will be sent for all leavings of subroutines even if several levels of subprograms do not contain any interpolation block information. Thus several SIMLeaveSubroutine calls can be sent successively.

### 3.11.13 Workpiece

#### Event

SIMNewWorkpiece	
(	
double	dVirtTime,
long	IChannel,
long	INumber,
VARIANT *	pvValue,
long	IVNCRunActionId
);	

#### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
INumber	Number of elements of the following array
pvValue	array of parameters of the new workpiece
IVNCRunActionId	IActionId of the preceding VNCRun()

## Description

The VNCK always sends this event when the program execution detects the activity of a new or changed workpiece statement in the IPO task. Actually simulation has to know the meaning and order of the parameters delivered.

---

### Note

There is an other version of this event:  
SIMNewWorkpieceSa ( ... );

---

## 3.11.14 Fixture

### Event

SIMNewFixture	
(	
double	dVirtTime,
long	IChannel,
long	INumber,
VARIANT *	pvValue,
long	IVNCRunActionId
);	

### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
INumber	Number of elements of the following array
pvValue	array of parameters of the new workpiece
IVNCRunActionId	IActionId of the preceding VNCRun()

## Description

The VNCK always sends this event when the program execution detects the activity of a new or changed fixture statement in the IPO task. Actually simulation has to know the meaning and order of the parameters delivered.

---

### Note

There is an other version of this event:  
SIMNewFixtureSa ( ... );

---

### 3.11.15 Fixed Stop and Measurement

#### Asynchronous service

```
VNCMotionContact
(
    long                                IChannel,
    VNCContactType_t                   tContactType,
    VNCProbeEdge_t                     tMeasProbeEdge,
    long                                INumber,
    long *                              plChanAxIndex,
    double *                            pdContactMcsAxVal,
    long                                IVNCMotionId,
    long                                plActionId
);

VNCMotionContactSetProbe
(
    VNCProbeEdge_t                     tMeasProbeEdge,
    long *                              plActionId
);
```

#### Event

```
SIMMotionContactEvent
(
    double                             dVirtTime,
    long                                IChannel,
    VNCContactAction_t                 tContactActionType,
    long                                INumber,
    long *                              plChanAxIndex,
    double *                            pdMcsAxValTarget,
    long                                IVNCMotionId,
    long                                IVNCRunActionId
);

SIMMotionContactResponse
(
    VNCResult_t*                       ptResult,
    long                                lActionId
);

SIMMotionContactSetProbeResponse
(
    VNCResult_t*                       ptResult,
    long                                lActionId
);
```

## Parameters

ptResult	Resulting value
dVirtTime	VNCK internal time stamp
IChannel	Channel number
tContactType	Type of executing the contact action
tMeasProbeEdge	Description of probe activity
INumber	Number of elements of the following arrays
plChanAxIndex	Array of channel axes indexes
pdContactMcsAxVal	Array of axes contact values
tContactActionType	Type of contact motion
pdMcsAxValTarget	Array of axes target values
IMotionId	MotionId of the contact motion action
IVNCRunActionId	Identifier from VNCRun()

## Description

The VNCK system will issue a SIMMotionContactEvent to report a programmed activity FXS or MEAS to be executed in the IPO task. This event describes as well the basic type of contact motion FXS or MEAS as the specific characteristics of this function. Measurement activities will furthermore describe the associated probe information. All the affected channel axes will be described with their motion target values.

---

### Note

Inside the SINUMERIK NCK FXS motions (drive to fixed stop) and MEAS motions (measuring) are handled differently. Before motion contact values for FXS motions become effective up to four IPO cycles will be necessary, whereas motion contact values for MEAS motions get effective immediately.

---

### Note

It is necessary to call VNCSetProgOutputOption() with the according flag VNC\_PROG\_OUTPUT\_CONTACT\_ACTION to let the VNCK system handle and issue any commands and events to contact motion activities.

---

Simulation can respond the SIMMotionContactEvent with a call of VNCMotionContact() to command how the contact action has to be performed by the VNCK system. There are several alternatives to execute the contact via parameter tContactType:

- VNC\_CONTACT\_TYPE\_INTERACTIVE: The VNCK system will perform the contact action immediately. If no axes are parameterized the axes positions will reach the values resulting from usual computing in VNCK kernel process from the actual time. If axes values are given by VNCMotionContact() the VNCK kernel will set the actual axes values to exactly these values independent from the values that have already been passed in program processing or not. Calling VNCMotionContact() can be done at any time during the execution of the block.

- VNC\_CONTACT\_TYPE\_PREDEFINED: The VNCK System will perform the contact action when one of the given channel axes reaches an actual position value given by simulation. Here the VNCK system stops further path interpolation of the actual block. VNCMotionContact() can be done at any time during the execution of the block.
- VNC\_CONTACT\_TYPE\_EXTERNAL: Using this mode simulation tells the VNCK system that all necessary activity to perform the motion contact will be done via an external way. This option is only used in combination with virtual PLC. VNCMotionContact() must be called before any further interpolation takes place.

---

**Note**

Actually the mode VNC\_CONTACT\_TYPE\_EXTERNAL can only be used for fixed stop motions.

---

---

**Note**

Actually only measuring functions MEAS or MEAW are supported.

---

SIMMotionContactResponse() will report whether the request can be handled.

After the contact action has been performed the VNCK system issues a further SIMMotionContactEvent() to finally report the result of handling the contact activity.

---

---

**Note**

Before VNCK system will report the performed contact at given contact positions there may be SIMPathOutputMCS() events reporting axes values beyond the contact positions. This is caused by the fact that SIMPathOutput() reports interpolation task command values whereas the contact check is computed on servo actual values. Nevertheless the command axes values will be updated by the actual values when finishing the block.

---

---

**Note**

There is a freezeMode VNC\_FREEZEMODE\_CONTACT\_MOTION to allow freeze interrupts after SIMMotionContactEvent().

---

Handling of initial states for measuring probes:

To change the initial state of probes there are further functions VNCMotionContactSetProbe() and the according SIMMotionContactSetProbeResponse(). The command function with its parameter tMeasProbeEdge can be used to set a probe to the state that the simulation system requires. E.g. this request is used after the probe is removed by following program lines from the contact point after the measurement has taken place.

#### Note

There are other versions of these functions:  
SIMMotionContactEventSa ( ... );  
VNCMotionContactSa ( ... );

#### Example for VNCK system controlled contact motion:

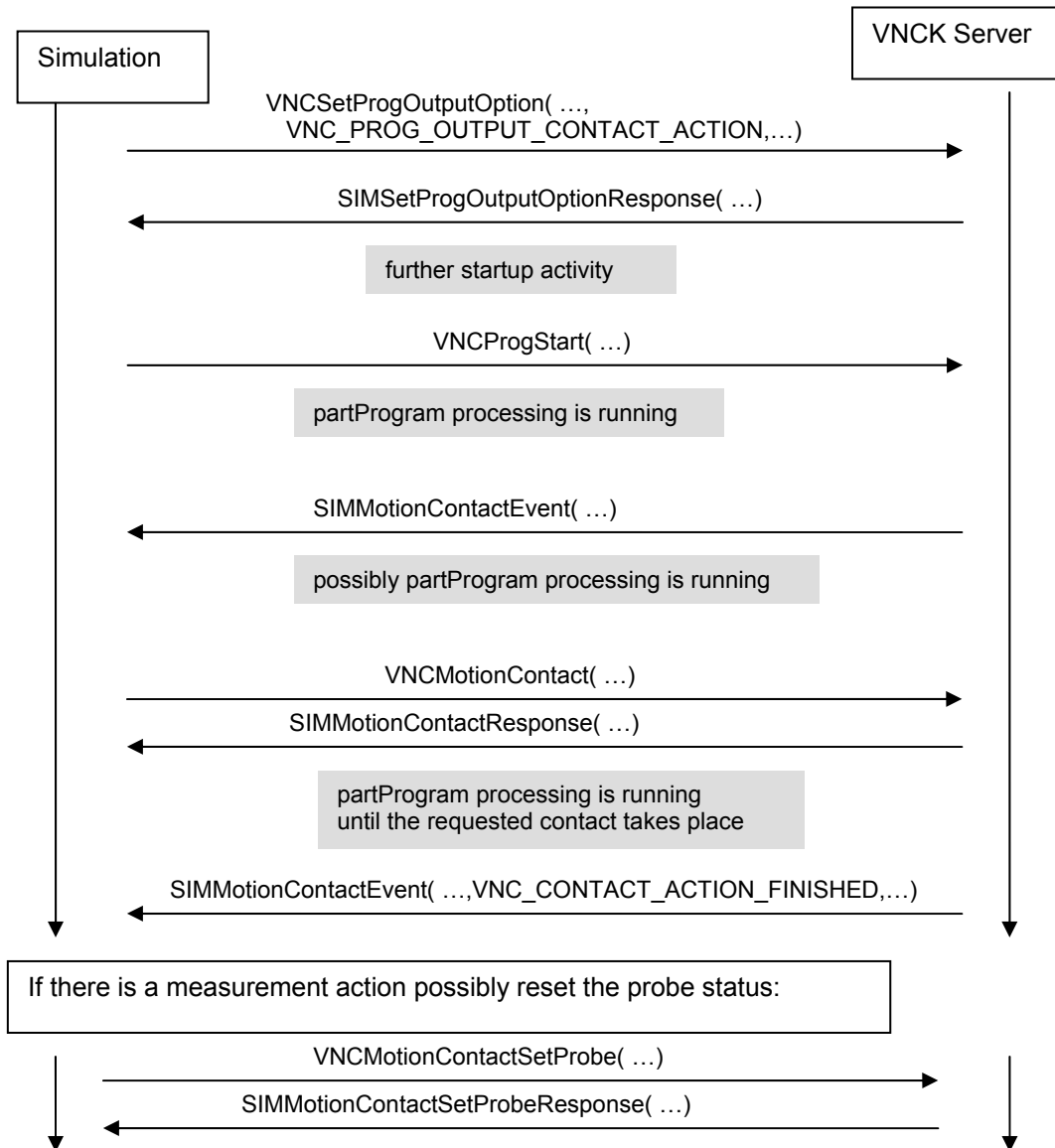


Figure 3-4: VNCK system controlled contact motion

## 3.12 Alarm Management

### 3.12.1 Alarm occurred

#### Events

```

SIMNewAlarmEvent
(
    double                dVirtTime,
    long                  IChannel,
    long                  IAlarmId,
    BSTR                  sAlarmText,
    VNCAAlarmQuitt_t      tAlarmQuitt,
    long                  IAlarmCookie
    long                  IVNCRunActionId
);

SIMNewAlarmDescription
(
    long                  IAlarmId,
    BSTR                  sAlarmText,
    VNCAAlarmQuitt_t      tAlarmQuitt,
    long                  IAlarmCookie
);

```

#### Parameters

dVirtTime	VNCK internal time stamp
IChannel	Channel number
IAlarmId	Alarm ID
sAlarmText	Description of alarm with parameters
tAlarmQuitt	Clear condition
IAlarmCookie	Identifier to the temporary alarm
IVNCRunActionId	IActionId of the preceding VNCRun()

#### Description

A SIMNewAlarmEvent is sent immediately from VNCK when the alarm occurs during the program process in VNCK. This event is described by the virtual time. The alarmId gives the alarm number. The alarm text describes the alarm itself with its VNCK alarm number, the channel in which the alarm occurred, the number of the program line corresponding to the alarm and further information on the alarm.

A SIMNewAlarm Description is sent when the HMI alarm server reports an alarm that is not yet reported by a SIMNewAlarmEvent.

Depending on the value of tAlarmQuitt specific actions will be necessary to quit alarms. Only for type VNC\_ALARM\_QUITT\_CANCEL an alarm can be canceled with VNCCancelAlarm(). The types VNC\_ALARM\_QUITT\_RESET and VNC\_ALARM\_QUITT\_REBOOT require a kernel reset or even a complete reboot of the VNCK system.

### 3.12.2 Alarm deleted

#### Event

```
SIMAlarmDeleted  
(  
    long                IAlarmId,  
    long                IAlarmEventId  
);
```

#### Parameters

IAlarmId	Alarm ID
IAlarmEventId	Identifier to the temporary alarm

#### Description

With this event the simulator is informed of the clearing of an alarm described by the actionId from the list of all active alarms in VNCK. The simulator can remove the alarm from a display eventually shown in its user interface.

---

#### Note

SIMAlarmDeleted(0, 0) informs the simulation that all actual alarms have been deleted (e.g. after a kernel reset). They might, however, be reissued via new SIMNewAlarmEvents.

Behaviour when BTSS connection fails:

If the BTSS connection between HMI software and NCK is aborted for any reason, for all alarm messages a SIMAlarmDeleted event will be issued. As soon as the BTSS connection is re-established SIMNewAlarmEvents will be sent for all active alarms. This behaviour is standard for real SINUMERIK controllers, but might be unwanted for simulation.

Adding the following lines to theVNC.ini will keep the BTSS connection alive also during freeze periods:

- [VNCK]
- activateBTSSwhileFreezing=1

This new setting is only available in VNCK4.5 or higher

---



### 3.12.3 Cancelling alarms

#### Synchronous service

```
VNCCancelAlarm  
(  
    long                IActionId  
);
```

#### Event

```
SIMCancelAlarmResponse  
(  
    VNCRResult_t*      ptResult,  
    long                IActionId  
);
```

#### Parameters

ptResult	Resulting value
IChannel	Channel number
IActionId	Identifier to alarm

#### Description

With this call the simulator can initiate canceling of all alarms in a VNCK channel with the appropriate cancel clearing condition. The response only provides information on the successful order of the command to the VNCK. The result of the canceling itself is reported by SIMAlarmDeleted events.

---

#### Note

VNCCancelAlarm must not be called more than one time when the NCK is in freeze mode. Before it is called a second time a VNCRun must be issued.

---

### 3.13 OEM Compile Cycles

Refer to the according paragraphs of the description of the VNCBoot() function.  
The bootOptionTypes VNC\_BOOTTYPE\_SIM\_DATA\_CC,  
VNC\_BOOTTYPE\_SRAM\_CC and VNC\_BOOTTYPE\_IBN\_CC are used for VNCK  
kernel loading and activating OEM cycles.

### 3.14 Extended Services

#### General Information

Extended services like block search and correction editor are not supported.

### 3.15 EXTCALL

You can set an entry in the initialization file 'theVNC.ini' to let the VNCK system  
automatically extend the search path for a part program stored on PC.

```
[DOMAIN]  
pcExtProgPathFile=< InstallPath>/sw/extProgPath.ini
```

For more information read the described extProgPath.ini file.

## 3.16 Reading / Writing values of Initial Parameters

### 3.16.1 General Information

VNCKServer can be parameterized to meet special needs.

The parameters (e.g. timeouts, directories, etc.) are initialized with values which are both hard coded, within the VNCK server source code, and stored in the following 2 files:

<InstallPath>\theVNC.ini

<AllUsersPath><sup>1</sup>\theVNC.ini

This last file is for user-defined values.

Moreover the user can define at runtime the actual value of parameters using the function: VNCSetIniParameter (...).

A parameter is identified by its 'Section' name and the 'Entry' name, which specifies its value in the 'theVNC.ini' file.

When VNCKServer reads an ini parameter, its value is retrieved according to the following rule:

- If a "user choice" has been imposed, through the function VNCSetIniParameter, that value is used (even if it is an empty string!).
- Otherwise a "standard choice" is done, i.e. the first not empty string found in the following sequence (in order of priority):
  1. the 'customer' value, defined through the function VNCSetIniParameter
  2. the user-defined value, present in the file <AllUsersPath>\theVNC.ini
  3. the predefined value, present in the file <InstallPath>\theVNC.ini
  4. the 'default' value, hard coded in the source code file.

Also the user can watch the value of a parameter, through the function: VNCGetIniParameter (...).

In the functions for reading/writing values of initial parameters, the argument 'VNCIniParamType\_t' specifies which one will be treated.

It can assume the following type, which have the described meaning:

VNC_INIPARAM_NONE	=	no choice
VNC_INIPARAM_DEFAULT	=	the source code
VNC_INIPARAM_THEVNCINI	=	the <InstallPath>\theVNC.ini
VNC_INIPARAM_USERTHEVNCINI	=	the <AllUsersPath>\theVNC.ini
VNC_INIPARAM_CUSTOMER	=	the run-time user-defined

---

<sup>1</sup> <AllUsersPath>

in Windows XP:

C:\Documents and Settings\All Users\Application Data\Siemens\Sinumerik\VNCK\v4.5

In Windows 7:

C:\Program Data\Siemens\Sinumerik\VNCK\v4.5

### 3.16.2 Writing the value of an initial parameter

#### Synchronous service

```
VNCSetIniParameter  
(  
    VNCIniParameterType_t      tIniParameterType,  
    BSTR                       sSectionName,  
    BSTR                       sEntryName,  
    BSTR                       sValue  
);
```

#### Parameters

tIniParameterType	specification on what value to consider
sSectionName	name of the section
sEntryName	name of the entry
sValue	value given to the parameter (string)

#### Description

Impose the use of a value for an initial parameter of the VNCK server.  
Synchronous service: the function returns the result and no response is sent.

Only if the first argument 'tIniParameterType' is 'VNC\_INIPARAM\_CUSTOMER', the given string becomes the run-time user-defined value. Otherwise the last argument 'sValue' has no meaning (and you can simply write "").

When the first argument 'tIniParameterType' is 'VNC\_INIPARAM\_NONE', then the run-time user-defined value is deleted, the "user-choice" is reset and the "standard choice" is restored.

### 3.16.3 Reading the value of an initial parameter

#### Synchronous service

```
VNCGetIniParameter
(
  VNCIniParameterType_t      tIniParameterType,
  BSTR                       sSectionName,
  BSTR                       sEntryName,
  BSTR                       * psvalue
);
```

#### Parameters

tIniParameterType	specification on what value to consider
sSectionName	name of the section
sEntryName	name of the entry
psvalue	pointer to the value returned (string)

#### Description

Retrieves the values of an initial parameter of the VNCK server.

Synchronous service: the function returns the result and no response is sent

When the first argument 'tIniParameterType' is 'VNC\_INIPARAM\_NONE', then the actual value (retrieved and used by the VNCKServer program in that moment) is returned.

---

#### Note

While VNCSetIniParameter only works for VNC\_INIPARAM\_CUSTOMER, VNCGetIniParameter is working for all enum types.

It might be confusing the VNCGetIniParameter delivers another value than the value which was set by VNCSetIniParameter if a different enum value is used.

---



## 4 VPLC Processing

The following chapters describe the API functions relating to the VNCK server as well as the event functions of the callback interface.

VPLC related server functions are marked by a prefix 'VPLC' (e.g. VPLCInitialize(...)). Callback functions are marked by a prefix 'SIM' (e.g. SIMVPLCHwConfigChanged(...)).

To use the IVPLC interface there is the precondition that a VPLC process was started in parallel with the VNCK kernel process.

---

### Note

VPLC is protected by a specific license option, which is not part of the standard VNCK license.

---



---

### Note

In fact, the most functions are implemented as a synchronous function. Therefore a warning value is returned to the requests to advise to this behavior.

---

### VPLC on multi-core operating systems:

By default the VPLC processes are running locked to core 0 (same as VNCK). Since VNCK4.5SP1 it is possible to define which core should be used for the VPLC processes by adding the following lines to theVNC.ini:

```
[VPLC]
VPLCCoreID=<IntegerValue>
```

The value <IntegerValue> starts with index 0, means core 0 is addressed.

---

### Note

This setting in theVNC.ini is only read from the theVNC.ini file in the VNCK installation folder.

---

## 4.1 VPLC Initializing and Shutdown

The following functions are needed to start and close handling of a virtual PLC as well as to initialize communication and scale the CPU usage of the VPLC process.

### 4.1.1 Establishing the Controller – Simulator Connection

#### Synchronous service

```
VPLCSetVPLCInterface  
(  
    ISIMVPLCCallback *      pSimVPLC  
);
```

#### Parameters

pSimVPLC	callback interface for VPLC events
----------	------------------------------------

#### Description

This function passes a pointer to an object in the simulator that receives the VNCK callbacks concerning VPLC handling. It is necessary to call this function as the very first call to the IVPLC interface to allow the VNCK server to fire asynchronous response events to simulation. Otherwise calls to the IVPLC interface will be rejected.

### 4.1.2 Initializing VPLC Handling

#### Synchronous service

```
VPLCInitialize ( );
```

#### Description

This function must be called next after VPLCSetVPLCInterface to let the server check license option, initialize communication with the VPLC process, retrieve the description of the VPLC IO hardware configuration and initialize further states.



### 4.1.3 Providing VPLC IO Hardware Configuration

#### Event

```
SIMVPLCHwConfigChanged  
(  
    long  
    VPLCHwConfig_t *          IResult,  
                                ptHwConfig  
);
```

#### Parameters

IResult	Resulting value of determining hwConfig
ptHwConfig	Pointer to memory describing the hwConfig

#### Description

This callback function informs simulation about the actual state of VPLC's hardware configuration description. The event is fired inside VPLCInitialize(). The parameter ptHwConfig is a void pointer. Using a compatible .h file simulation must be able to cast an according data type pointer to access the description.

### 4.1.4 Terminating VPLC Handling

#### Synchronous service

```
VPLCTerminate ( );
```

#### Description

This function stops the VNCK servers management of handling a VPLC.

### 4.1.5 Controlling CPU usage of the VPLC process

#### Synchronous service

```
VPLCSetCpuScale  
(  
    long *          plScale  
);
```

#### Description

This function orders the VNCK system to limit the CPU usage of the VPLC process.  
The parameter unit is used as CPU percentage. It works as input for the requested value as well as output for the by the VPLC process determined value.

## 4.2 VPLC Leds and Switches

The following functions can be used to watch and control the VPLC state.

### 4.2.1 Reading VPLC Operation State

#### Synchronous Service

```
VPLCGetLeds
(
    VPLCLeds_t *          ptLeds
);
```

#### Parameters

ptLeds	Array of states
--------	-----------------

#### Description

This function returns the on/off or flushing states of the VPLC operation and error conditions.

### 4.2.2 Watching VPLC Operation State

#### Synchronous Services

```
VPLCWatchLeds
(
    long *                  plActionId
);

VPLCUnWatchLeds
(
    long                    lActionId
);
```

#### Event

```
SIMVPLCWatchLedsEvent
(
    long                    IResult,
    VPLCLeds_t *          ptLeds,
    long                    lActionId
);
```

#### Parameters

IResult	Resulting value of watch event
ptLeds	Array of VPLC conditions
[p]lActionId	Identifier to callback functions

## Description

Using the service functions simulation can order the VNCK system to start or stop sending as well an initial event as continuous events reporting the actual status and changing states of the operation and error conditions of the VPLC. Whenever a status is changing the event is fired.

### 4.2.3 Setting VPLC Switch

#### Synchronous Service

```
VPLCSetSwitch  
(  
    VPLCSwitchType_t          tSwitch  
);
```

#### Parameters

tSwitch	VPLC switch to be activated
Description	
This function activates the described switch of the VPLC.	

### 4.2.4 Reading VPLC Switch

#### Synchronous Service

```
VPLCGetSwitch  
(  
    VPLCSwitchType_t *      ptSwitch  
);
```

#### Parameters

ptSwitch	Actual activated VPLC switch
Description	
This function reports the actual activated switch of the VPLC.	

### 4.2.5 Watching VPLC Operation States

#### Synchronous Services

```
VPLCWatchSwitches
(
    long *                                plActionId
);

VPLCUnWatchLeds
(
    long                                lActionId
);
```

#### Event

```
SIMVPLCWatchSwitchesEvent
(
    long                                IResult,
    VPLCSwitchType_t                  tSwitch,
    long                                lActionId
);
```

#### Parameters:

IResult	Resulting value of watch event
tSwitch	Actual activated VPLC switch
[p]lActionId	Identifier to callback functions

#### Description

Using the service functions simulation can order the VNCK system to start or stop sending as well an initial event as continuous events reporting the actual activated switch of the VPLC. Whenever the VPLC status is changing the event is fired.

## 4.3 VPLC Progress Control

The following functions allow synchronizing the simulation process with the VPLC by setting and handling time slices.

### 4.3.1 Activating VPLC Synchronisation

#### Synchronous service

```
VPLCRun  
(  
    long *                               plActionId  
);
```

#### Parameter

plActionId	Identifier to callback function
------------	---------------------------------

#### Description

This service causes the VNCK system to watch the synchronization between the VNCK kernel and the VPLC process. For every cycle of the synchronization the call-back function SIMVPLCFreeze will be called. This tells simulation that the VNCK system resides in a consistent data status. Now simulation can access the given data IO interface to read and write VPLC IO data. After the VPLCFreeze this function must be called to continue processing.

### 4.3.2 Deactivating VPLC Synchronisation

#### Synchronous service

```
VPLCIdle  
(  
    long *                               plActionId  
);
```

#### Parameter

plActionId	Identifier to callback function
------------	---------------------------------

#### Description

This service causes the VNCK system to stop notifying simulation in every synchronization cycle. Nevertheless, the VNCK kernel and the VPLC process will continue working synchronized. VNCK server quits sending SIMFreeze and will not expect any more VPLCRun commands.

**4.3.3 VPLC Freeze**

**Event**

SIMVPLCFreeze	
(	
double	dVirtTime,
long	IVPLCRunActionId
);	

**Parameter**

IResult	Resulting value
dCycleTime	Value describing the active time
[p]IActionId	Identifier to callback function

**Description**

For every cycle of the synchronization this call-back function will be called. It tells simulation that the VNCK system resides in a consistent data status. Now simulation can access the given data IO interface to read and write VPLC IO data.

Simulation must call VPLCRun to release the VNCK system for the next loop.

## 5 NCU Link Processing

The following chapters describe the API functions relating to the VNCK server as well as the event functions of the callback interface for simulating SINUMERIK systems with NCU link.

VNCK server functions are marked by a prefix 'VNCLink' (e.g. VNCLinkInitialize(...)). Callback functions are marked by a prefix 'SIMLink' (e.g. SIMLinkInitializeResponse(...)).

---

**Note**

NCU Link is protected by a specific license option, which is not part of the standard VNCK license.

---

---

**Note**

The NCU Link interface of VNCK is designed to work with two NCU units, which run in a synchronized fashion also in real world. This requires special link settings. These settings are part of the machine data settings when creating series commissioning files from the real controller.

---

## 5.1 Link System Initializing and Shutdown

The following functions are needed to start and close handling of a NCU link system describing several NCU units working NCU link synchronized. Nevertheless the link interfaces can be used to control one NCU unit using the link interface functions.

### 5.1.1 Establishing the Controller – Simulator Connection

#### Synchronous service

```
VNCLinkSetSIMInterface  
(  
    ISIMLinkCallback *      pSimLink,  
    ISIMCallbackLicense *   pSimLicense,  
    BSTR                    sLinkName  
);
```

#### Parameters

pSimLink	callback interface for link events
pSimLicense	callback interface for license management
sLinkName	name of the link unit

#### Description

This function passes a pointer to an object in the simulator that receives the VNCK callbacks concerning link handling. It is necessary to call this function as the very first call to the VNCK system interface to allow the VNCK server to send asynchronous response events to simulation. Otherwise calls to the VPLC system will be rejected. Furthermore there is a pointer to a simulation callback object that handles license requirements if the VNCK system is designed as an ISV (independent software vendor) system. To identify the actual link unit there is a parameter sLinkName.

---

#### Note

In fact, this function is implemented as a synchronous function. Therefore a warning value is returned to the request to advise to this behavior.

---

---

#### Note

The SAVE option for single NCUs is not supported when working with NCU-Link.

---



### 5.1.2 Initializing the Link System

#### Asynchronous service

```

VNCLinkInitialize
(
    long                                INumNcu,
    long *                             pLinkInitActionId
);

```

#### Event

```

SIMLinkInitializeResponse
(
    VNCRResult_t *                     ptResult,
    long                               ILinkInitActionId
);

```

#### Parameters

ptResult	result value
INumNcu	number of NCU units
[p]ILinkInitActionId	Identifier to callback function

#### Description

This function must follow VNCLinkSetSIMInterface to define the number of NCU units to be synchronized inside the given link unit. The callback event will be sent when all the NCU units were created by following VNCLinkCreateNcu calls.

### 5.1.3 Defining the NCU Units

#### Synchronous service

```
VNCLinkCreateNcu  
(  
  BSTR  
  IVNCNcuServer * *          sNcuName,  
                              ppVncNcu  
);
```

#### Parameters

sNcuName	name of the NCU unit
ppVncNcu	pointer to IVNCNcuServer object

#### Description

After VNCLinkInitialize all NCU units must be identified with a NCU name. The VNCK system now instantiates IVNCNcuServer objects that will be used to control the NCU units regarding the NCU local functions.

---

#### Note

In fact, this function is implemented as a synchronous function. Therefore a warning value is returned to the request to advise to this behavior.

---

### 5.1.4 Controlling the NCU Startups

#### Asynchronous service

```
VNCLinkBoot
(
    long *                                pLinkBootActId
);
```

#### Event

```
SIMLinkBootResponse
(
    VNCRResult_t *                        ptResult,
    long                                lLinkBootActId
);
```

#### Parameters

ptResult	result value
[p]lLinkBootActId	Identifier to callback function

#### Description

Using this function the sequence of booting all the NCU units is started. When all the NCU units are running simulation will be notified by the callback function.

---

#### Note

When working with Linked SRAM files it must be ensured, that Link-SRAMs, which were created based on HMIBase are only booted with HMIBase activated. The same applies for Link-SRAMs, which were created based on Operate services. These must be booted only with activated Operate services.

---

### 5.1.5 Setting the Link State

#### Asynchronous service

```
VNCLinkSetState
(
  VNCTBooleanType_t          tLinkStateActive,
  long *                      pLinkSetStateActId
);
```

#### Event

```
SIMLinkSetStateResponse
(
  VNCTResult_t *             ptResult,
  long                      lLinkSetStateActionId
);
```

#### Parameters

ptResult	result value
tLinkStateActive	required link state
[p] lLinkSetStateActionId	Identifier to callback function

#### Description

Using this function simulation can change the link state of the booted NCU units. That means the NCU units can be set into a time synchronized mode where all the NCUs are working on the same virtual time axis. Also the NCU link can be released by this function. For both modes the VNCK server will reboot the NCU units with an according environment. SimKernelResetEvents will be fired by each NCU unit. The callback function SIMLinkSetStateResponse will notify the end of server activities to change the state of the link unit.

### 5.1.6 Terminating a Link Session

#### Asynchronous service

```

VNCLinkShutdown
(
    long *                                pLinkShutdownActionId
);

```

#### Event

```

SIMLinkShutdownResponse
(
    VNCRResult_t *                        ptResult,
    long                                ILinkShutdownActionId
);

```

#### Parameters

ptResult	result value
[p] ILinkShutdownActionId	Identifier to callback function

#### Description

Using this function the VNCK server is shutting down all NCU units defined within the link unit. At last the infrastructure of the link unit itself is cleaned up and released.

---

#### Note

The NCU units cannot be released with the IVNCServer function VNCSShutdown anymore.

---



---

#### Note

After calling VNCLinkShutdown the VNCK COM interface must be re-initialized by calling VNCLinkSetSIMInterface before calling the next VNCLinkBoot.

---

### 5.1.7 Link System Startup via a VMF

#### Asynchronous service

```

VNCLinkSetup
(
    BSTR                                sLinkVmfDataPath,
    ISimLinkNcuFactory*                pSimLinkNcuFactory,
    long *                             plLinkSetupActionId
);

```

#### Event

```

SIMLinkSetupResponse
(
    VNCResult_t *                      ptResult,
    long                               lLinkSetupActionId
);

```

#### Parameters

ptResult	result value
sLinkVmfDataPath	VmfFile of link unit
pSimLinkNcuFactory	Simulation's callback factory
[p] lLinkSetupActionId	Identifier to callback function

#### Description

This is a more convenient startup function that includes all the previously described functions for activating a link unit system until part programs can be run. Simulation must provide a vmfFile that contains all the descriptions as well of the link unit as of the NCU units to run under synchronized link mode. This vmfFile can be created with the interface function VNCLinkSaveData.

Furthermore simulation must provide a pointer to a callback factory object that allows the VNCK server to ask for all the required callback objects of the included NCU units.

## 5.2 Link System Progress Controlling

The following functions are used to let the NCU units process part programs inside their channels under a common slice control. All NCU units will be synchronized on the same virtual time bar.

### 5.2.1 Setting Link Slices

#### Asynchronous services

```

VNCLinkSetTimeSlice
(
    double                                dSliceIpoTime,
    long *                                pLinkSetTimeSliceActionId
);

VNCLinkSetSliceMode
(
    VNCSliceMode_t                       tSliceMode,
    VNCSliceValues_t *                   ptSliceValues,
    long *                                pLinkSetSliceModeActionId
);

```

#### Events

```

SIMLinkSetTimeSliceResponse
(
    VNCRResult_t *                       ptResult,
    long                                  lLinkSetTimeSliceActionId
);

SIMLinkSetSliceModeResponse
(
    VNCRResult_t *                       ptResult,
    long                                  lLinkSetSliceModeActionId
);

```

#### Parameters

ptResult	result value
dSliceIpoTime	value for linked time slice
tSliceMode	mode for general link slices
ptSliceValues	values for general link slices
[p]lLinkSetTimeSliceActionId	Identifier to callback function
[p]lLinkSetSliceModeActionId	Identifier to callback function

#### Description

Using VNCLinkSetTimeSlice simulation can parameterize a link global time slice for all the NCU units. The function VNCLinkSetSliceMode allows setting all the other slice modes including their criterion values for all the NCU units by one call.

---

**Note**

Since this value must be unique to all NCU units inside a link unit the IVNCServer function VNCSetSliceMode is no longer allowed to be called inside a link unit. Nevertheless the values except the time slice can be set individually via the IVNCNcuServer interface.

---

## 5.2.2 Processing the next Slice

### Synchronous service

```
VNCLinkRun  
(  
    long *                                pLinkRunActionId  
);
```

### Parameters

pLinkRunActionId	Identifier to SIMLinkFreeze callback function
------------------	---

### Description

This function is used to let the VNCK system process the next slice. The function is necessary after a SIMLinkFreeze event that idled all link synchronized the NCU units at the same virtual time. Calling this function let all NCU units proceed their part program processing up to the next slice or freeze criterium becomes true.

---

**Note**

In fact, this function is implemented as a synchronous function. Therefore a warning value is returned to the request to advise to this behavior.

---



### 5.2.3 Link Freeze

#### Event

```
SIMLinkFreeze
(
    double                dVirtTime,
    VNCFreezeReason_t    tLinkFreezeReason,
    long                 lLinkRunActionId
);
```

#### Parameters

dVirtTime	Virtual time stamp
lLinkFreezeReason	Reasons for freezing
lLinkRunActionId	Identifier to VNCLinkRun function

#### Description

This event is fired from the VNCK system when all NCU units are residing idled at the same given virtual time. The parameter lLinkFreezeReason describes why there is a break in processing part programs. If there are several and / or different reasons from one or several NCU units the individual freezeReasons are combined into one parameter of SIMLinkFreeze. All the NCU units have sent a SIMNcuFreezeInfo event previously to notify their local freezeReasons.

After sending SIMLinkFreeze the VNCK will no longer consumes virtual time. The VNCK usually changes to this state if any of the slice or freeze conditions of one of the NCU units are met.

---

#### Note

Since there must be one unique event notifying the freeze state of the link unit the callback events SIMFreeze from ISIMCallback will not be fired.

---

## 5.3 Link System Status Saving and Resetting

The following functions are used to store and reset the status of complete NCU link systems.

### 5.3.1 Saving the States of the Link Components

#### Asynchronous service

```
VNCLinkSaveData  
(  
  VNCSaveData_t          tSaveMask,  
  BSTR                   sVmfileName,  
  long *                  plLinkSaveDataActionId  
);
```

#### Event

```
SIMLinkSaveDataResponse  
(  
  VNCRresult_t *          ptResult,  
  long                    lLinkSaveDataActionId  
);
```

#### Parameters

ptResult	result value
tSaveMask	mode for save operation
sVmfileName	name of file containing saved data
[p]plLinkSaveDataActionId	Identifier to callback function

#### Description

The actual state of the complete VNCK or single parts of data of the simulation machine will be saved for future startups or updates of link units and their included NCU units. The parameter tSaveMask describes which data of the actual running VNCK system has to be stored.

VNC\_SAVEDATA\_SRAM:

Using this value either the complete state of the last shut down VNCK kernel or the state of the actual running VNCK kernel is stored to a file 'vmfSim.dat'. You can use this file for a VNCLinkSetup() call.

---

#### Note

Up to now only the save mode VNC\_SAVEDATA\_SRAM is possible.

When there is a link unit the usage of the save mode VNC\_SAVEDATA\_SRAM is no longer possible by the IVNCServer function VNCSaveData. Only VNCLink-SaveData can create Vmf files of link systems.

---

## 5.3.2 Refreshing the States of the Link Components

### Asynchronous service

```
VNCLinkMatchData
(
  VNCLinkMatchData_t      tRefreshMask,
  long *                   pLinkMatchDataActionId
);
```

### Event

```
SIMLinkMatchDataResponse
(
  VNCLinkMatchData_t *    ptResult,
  long                    lLinkMatchDataActionId
);
```

### Parameters

ptResult	result value
tRefreshMask	mode for refresh operation
[p]LinkMatchDataActionId	Identifier to callback function

### Description

Using VNCLinkMatchData a link unit will be updated by the dates and values of the described machine. Depending on the mask different dates, i.e. machine data, tool data, guides, cycles types will be attached

VNC\_MATCHDATA\_SRAM:

Using this value the complete state of the running VNCK is updated from a stored SRAM file 'vmfSim.dat'. This method prevents simulation from shutting down and rebooting a VNCK system to restore a VNCK state.

---

#### Note

Until now only the value VNC\_MATCHDATA\_SRAM can be used.

---

## 5.4 Link NCU Management

When there is a link system some of the standard IVNCServer interface functions are not available. This is because these functions must be executed under the control of the link management. The functions on the IVNCServer interface will return a matching error value. Furthermore some functions are necessary to handle link NCU units. Therefore there is an interface IVNCNcuServer that is derived from the standard IVNCServer interface.

### 5.4.1 Requiring Simulation Callback Objects

#### Event

```
CreateNcu
(
    BSTR                                sLinkName,
    BSTR                                sNcuName,
    IVNCNcuServer*                      pNcuServer,
    ISIMCallback**                      ppSim,
    ISIMNcuCallback**                  ppSimNcu,
    ISIMCallback_ext**                  ppSimNcuExt,
    ISIMCallback_sa**                  ppSimNcuSA
);
```

#### Parameters

sLinkName	name of the link unit
sNcuName	name of the NCU unit
pNcuServer	pointer to NCU server object
ppSim	pointer for basic callback interface
ppSimNcu	pointer for NCU server callback interface
ppSimNcuExt	pointer for extended callback interface
ppSimNcuSA	pointer for safeArray function callback interface

#### Description

When the VNCK system is starting up a link unit based on a VNCLinkSetup call the VNCK server will require pointers to callback objects from simulation. These callback objects are necessary to handle the range of events that a call VNCSetsimInterface allows for single NCU units. Furthermore the event delivers a pointer to a IVNCNcuServer object to simulation that must be used to address function calls to the named NCU unit. This interface must be used instead of the single NCUs interface IVNCServer.

## 5.4.2 Establishing the Link NCU Controller – Simulator Connection

### Synchronous service

```
VNCNcuSetSIMInterface
(
    ISIMCallback *           pSim,
    ISIMNcuCallback *        pSimNcu,
    ISIMCallback_ext *       pSim_ext,
    ISIMCallback_sa *        pSim_sa
);
```

### Parameters

pSim	basic callback interface
pSimNcu	NCU server callback interface
pSimNcuExt	extended callback interface
pSimNcuSA	safeArray function callback interface

### Description

After calling VNCLinkBoot in the sequence of starting up a link unit simulation must boot the NCU units. Analogous to a single NCU unit this function delivers pointers to objects in the simulator that receive the VNCK callbacks. At the very least it is necessary to send the interface pointer to the object handling the basic and the NCU server functions. The other pointers refer to objects accepting callbacks for the extended scope of functions and / or functions using safeArrays for transferring lists of data. VNCNcuSetSimInterface () must be called before any other IVNCNcuServer function call in order to be able to receive callbacks from the VNCK.

### 5.4.3 Link NCU Controller start-up

#### Asynchronous service

```
VNCNcuBoot
(
    BSTR                    sLinkName,
    BSTR                    sNcuName,
    VNCBootType_t          tBootType,
    BSTR                    sBootDataPath,
    long *                  plBootActionId
);
```

#### Event

```
SIMNcuBootResponse
(
    VNCRResult_t *          ptResult,
    BSTR                    sNcuName,
    long                    lBootActionId
);
```

#### Parameters

ptResult	result of the NCU boot call
sLinkName	name of the link unit
sNcuName	name of the NCU unit
tBootType	boot type of initialization data
sBootDataPath	path to initialization data
[p]lBootActionId	Identifier to callback function

#### Description

Using this function simulation starts up a NCU unit inside a link unit. For the description of the parameters tBootType and sBootDataPath please look for the reference of VNCBoot inside the IVNCServer interface.

---

#### Note

The callback event SIMNcuBootResponse is also used when starting up a link system via VNCLinkSetup. In these cases simulation is notified about the state of starting the NCU units.

---

## 5.4.4 Setting Link NCU Slice Mode

### Asynchronous service

```
VNCNcuSetSliceMode
(
  VNCSliceMode_t          tSliceMode,
  VNCSliceValuesNcu_t *   ptSliceValues,
  long *                   plActionId
);
```

### Parameters

tSliceMode	Pattern describing the active slice modes
ptSliceValues	Array for options dependent on values

### Description

This function is analogous to VNCSetSliceMode from IVNCServer. The difference consists of that a time slice cannot be set since all the NCU units of the link unit must observe the same time slice value. All the other slice options can be chosen individually to each NCU unit.

---

#### Note

There is no ISIMNcuSetSliceModeResponse event. For the asynchronous response event the function ISIMSetSliceModeResponse from the ISIMCallback interface is used.

---

## 5.4.5 Link NCU Controller Shutdown

### Event

```
SIMNcuShutdownResponse
(
  VNCRresult_t *          ptResult,
  BSTR                    sNcuName,
  long                    IShutdownActionId
);
```

### Parameters

ptResult	Resulting value of NCU shut down
sNcuName	Name of the NCU unit shutted down
IShutdownActionId	Identifier to service IVNCLinkShutdown

### Description

This event is issued in sequence of a IVNCLinkShutdown request to notify simulation about the state of shutting down the single NCU units of a link unit.

### 5.4.6 Link NCU Controller Freeze

#### Event

```
SIMNcuFreezeInfo  
(  
    double                dVirtTime,  
    VNCFreezeReason_t    tNcuFreezeReason,  
    long                  lLinkRunActionId  
);
```

#### Parameters

dVirtTime	Virtual time stamp
tNcuFreezeReason	Reasons for freeze
lLinkRunActionId	lActionId of the preceding VNCLinkRun

#### Description

The VNCK system uses this function to inform the simulator of each NCU local reasons before sending a SIMLinkFreeze event. Thus simulation gets knowledge about the freeze reason of each NCU unit before the linkSlice closing event SIMLinkFreeze is sent.

---

#### Note

This is only an additional info event. Simulation has to wait for SIMLinkFreeze before the link unit can be regarded as frozen.

---



## 6 VNCK License

To be able to use VNCK it is necessary to be a registered VNCK user.

Two license options are available:

- Licensing via ISV license model  
The VNCK license is coupled to the license model of the ISV software.  
The ISV has to comply with Siemens licensing standards.
- Licensing via USB dongle  
The VNCK license is granted through a USB dongle which contains one individual license each.

For registration as a new VNCK customer please contact your local Siemens sales agent.

### 6.1 ISV License Checking

#### Event

```

SIMEncrypt
(
  BSTR
  long *
  BSTR
  long
  long *
);
sBufferCryptFile,
pLenCrypt,
sBufferFile,
ILen,
pResult

```

#### Parameters

sBufferCryptFile	File containing the encrypted data stream
pLenCrypt	Length of encrypted data stream
sBufferFile	File containing data stream to be encrypted
ILen	Length of data stream to be encrypted
pResult	Result of simulations encryption

## Description

If the simulation system is using the Siemens ISV License System to identify itself as a registered user this function asks the simulation system by this callback interface function to encrypt a given file content. The VNCK system examines the result to allow the VNCK system to run or not.

To do this there are different ways supported by software the VNCK provides at installation time. The files are stored under the directory `vnck/sw/license`. There are code template files `EPC_VC.cpp` and `EPC_VB.frm` that show how to implement the `SIMEncrypt` callback.

## 7 More General Information

### 7.1 Preparation of the HMI Base System

By default simulation does not need to assign a name to the NCU that will be used when booting a virtual NCU. But either if there is a not empty name parameter to the VNCBoot() command of a single NCU system or if there is a link system simulation must adapt some entries in HMI Base server ini-files and register some prepared data access services.

For changing the environment the following string 'MY\_NCU' should be substituted by the name simulation wants to use for the interface functions of the actual NCU. By default the VNCK system is installed with a name 'VNCK'. Simulation has not to do any changes if no name is used at the interface.

After executing all the following steps reboot the PC to activate the changes!

#### 7.1.1 Enabling the OPC Data Access

Copy the directory <InstallPath>\HMIBase\dataaccessTemplate to a new directory <AllUsersPath>\HMIBase\dataaccess\_MY\_NCU. Then edit the file SOPSRVR.ini:

```
edit ProgId = OPC.SINUMERIK.MY_NCU
create a new GUID (e.g. using GUIDGEN.EXE provided by Microsoft)
edit ClassId = {new GUID}
edit SymbolicName = OPC.SINUMERIK.MY_NCU
edit RegKey = SINUMERIK.MY_NCU
edit IVarServer = @MCVar.Var2:MY_NCU
```

A valid GUID can be obtained by an appropriate system call to the Windows operating system.

Then execute on a command shell:

```
"<AllUsersPath>"1\HMIBase\dataaccess_MY_NCU\SOPC_MachineSwitch.exe
/regserver".
```

---

<sup>1</sup> AllUsersPath>

in Windows XP:

C:\Documents and Settings\All Users\Application Data\Siemens\Sinumerik\VNCK\v4.5

In Windows 7:

C:\Program Data\Siemens\Sinumerik\VNCK\v4.5

### 7.1.2 Single NCU Setting

Look for <HMIInstallPath><sup>2</sup>/user/mmc.ini:

```
[GLOBAL]
NcddeMachineName= MY_NCU
NcddeDefaultMachineName= MY_NCU
NcddeMachineNames= MY_NCU

[VNCK]
ADDRESS0=3,LINE=10,NAME=/NC,PROFILE= MY_NCU_COS_HMI_L4_INT
ADDRESS1=
vnckMachine=1
```

If there are several single NCUs to be run there must be the following entries, where MY\_NCU1 is used as an arbitrary default:

```
[GLOBAL]
NcddeMachineName= MY_NCU1
NcddeDefaultMachineName= MY_NCU1
NcddeMachineNames= MY_NCU1, MY_NCU2, MY_NCU3

[MY_NCU1]
ADDRESS0=3,LINE=11,NAME=/NC,PROFILE= MY_NCU1_COS_HMI_L4_INT
ADDRESS1=
vnckMachine=1

[MY_NCU2]
ADDRESS0=3,LINE=12,NAME=/NC,PROFILE= MY_NCU2_COS_HMI_L4_INT
ADDRESS1=
vnckMachine=1

[MY_NCU3]
ADDRESS0=3,LINE=13,NAME=/NC,PROFILE= MY_NCU3_COS_HMI_L4_INT
ADDRESS1=
vnckMachine=1
```

The steps for enabling the OPC Data Server must be performed to all requested NCUs.

---

<sup>2</sup> <HMIInstallPath>  
C:\HMIAdv

### 7.1.3 Link NCU Setting

Look for <HMIInstallPath>/user/mmc.ini:

```
[GLOBAL]
NcddeDefaultMachineName= net:MY_NCU1
NcddeMachineNames=net
```

All NCUs working in the link unit must be parameterized by the file 'netnames.ini' that must be provided by simulation (see the reference books of Siemens SINUMERIK 840D sl HMI for more information). Copy the file 'netnames.ini' to the directory .../<HMIInstallPath>/user .

The steps for enabling the OPC Data Server must be performed to all requested NCUs.

---

**Note**

Use the name 'net:MY\_NCUx' at all the relevant places in the ini file.

---

## 7.2 Languages

All alarm texts are reported in English. Nevertheless the simulator can change the language either to German, Spanish, French or Italian.

If this is required, open the file 'mmc.ini' in the directory 'user' stored parallel to the 'vnck' directory. Set the entry 'Language' to the initials of your selected language. The possible initials are listed in the commented entry 'LanguageList'.

```
[LANGUAGE]
Language=UK
;LanguageFont=Europe
;LanguageList=GR, UK, SP, FR, IT
```

Remember not to change any other lines in the 'mmc.ini' file.

